Metastorm BPM®
Release 7.6

Designer User Manual

May 2008

Metastorm Inc.
email: inquiries@metastorm.com
http://www.metastorm.com

Copyrights and Trademarks

© 1996-2008 Metastorm Inc. All Rights Reserved.

Copyright Notice

- Metastorm®, Metastorm BPM®, e-Work®, Process Pod®, Enterprise Process Advantage®, ProVision®, The Best Process Wins®, Proforma®, Metastorm Knowledge Exchange™, Metastorm DNA™, Metastorm Discovery™, STAR™, Insight™, Envision™, and Metastorm Enterprise™ are either registered trademarks or trademarks of Metastorm in the United States and/or other countries.
- Microsoft®, Outlook®, SQL Server™, Windows®, Vista®, Active Directory®, Visual Basic®, JScript®, SharePoint® and BizTalk® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Adobe® is a registered trademark of Adobe Systems, Inc.
- Netscape® is a registered trademark of Netscape Communications Corporation.
- Other trademarks are the property of their respective owners.

Disclaimer

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Metastorm accepts no responsibility, and offers no warranty whether expressed or implied, for the accuracy of this publication.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the express written permission of Metastorm Inc.

The information in this document is subject to change without notice.

Metastorm Inc.

email: inquiries@metastorm.com

http://www.metastorm.com

Metastorm BPM Release 7.6

Designer User Manual

Table of Contents

1 GETTING STARTED 16

1.1 Metastorm BPM Components 17

1.2 Design Basics 18

1.2.1 Concepts and Vocabulary 18

1.2.2 Designing a Procedure 21

Define the Procedure Properties 21
Define the Initial Action 23
Define and Prioritize Tasks 23
Identify Participants 23
Determine Necessary Information 24
Determine Possible Outcomes 24
Identify Reusable Elements 25
Run a Simulation of the Procedure and Analyze the Data 25
Defining Simulation Data 25
Maintain the Procedure 25

1.3 The Designer Interface 27

1.3.1 Title Bar 28

1.3.2 Menus 28

The File Menu 28
The Edit Menu 29
The View Menu 30
The Component Menu 31
The Help Menu 31
The Tools Menu 31
Pop-up Menus 32
Dockable Menus 32
Personalized Menus 33

1.3.3 Toolbars 34

General Toolbar 35
Map toolbar and Form toolbar 36

1.3.4 Procedure Explorer 38

1.3.5 Component Bar 40

1.3.6 Main Pane 42

1.3.7 Status Bar 42

1.3.8 Properties editor 42

1.4 Saving, Publishing, and Retrieving Procedures 44

1.4.1 File Locations 44

1.4.2 File Types 44

1.4.3 Saving a Procedure 44

1.4.4 Exporting a Procedure or Map 45

1.4.5 Opening a Procedure 45

1.4.6 Importing a Procedure or Map *46*
1.4.7 Validating a Procedure *46*
1.4.8 Publishing a Procedure *46*
Data Formats 48
1.4.9 Retrieving a Published Procedure or Library *48*

2 LIBRARIES 50

2.1 Creating a Library 50
2.1.1 Creating a new library *51*
2.1.2 Adding library parts *52*
Forms 52
Form Segments 52
Map Segments 52
Scripts 53
LDAP Alias 53
Integration Wizard Collection 53
Flags 54
2.1.3 Saving and publishing a new library *54*
2.2 Associating a Library with a Procedure 54
2.2.1 Using a library *54*
2.2.2 Accessing a library's forms, form segments, map segments and scripts *56*
2.2.3 Inheritance of Roles from Libraries *56*
Nested Map Segments 57
Recursive Map Segments 57
Duplicate Names 57
Variables 58
Removing Libraries 58
Role Maintenance 58
2.2.4 Editing a Library *58*
2.2.5 Accessing a library's LDAP Alias *58*
2.2.6 Accessing a library's Integration Wizard collection *58*
2.3 Working with a Procedure that uses a Library 59
2.3.1 Library Versions *59*
2.3.2 Loading the required version of the library *59*
Summary 62
2.3.3 Reverting to an Earlier Library Version *63*
2.3.4 Working Offline *63*
To take a library offline 64
To take a library online 64
Updating the library offline 65

3 MAPS 66

3.1 Creating a New Map 67
3.1.1 Working with Map Elements *67*
Adding Stages to a Map 67
Adding Actions to a Map 67
Adding Comments to a Map 68
Moving Map Elements 68
Moving Map Elements 69
Copying and Pasting Map Elements 69

 © Metastorm Inc., 2008

Determining Map Element Alignment 69
Determining Map Element Layering 69
Removing Elements from a Map 70
3.2 Properties editor 70
3.2.1 The Start Box 72
Creation Actions 72
3.2.2 Defining Map Properties 72
Defining a Map's Map Properties 72
Defining a Map's Stages Order Properties 74
Defining a Map's Notes Properties 75
3.3 Stages 75
3.3.1 Stage-independent Tabs 77
Defining Stage Properties 77
Defining Roles Properties 79
Defining Do This Properties 80
Defining Notes Properties 82
3.3.2 Stage-specific Tabs 82
Defining a Sub-procedure Stage's Maps Properties 82
Defining a Common Stage's Applied to Properties 83
3.3.3 Renaming and Deleting Stages 83
3.4 Actions 83
3.4.1 Working with Actions 85
Adding Actions to a Map 85
Deleting Actions from a Map 86
Moving Actions 86
Conditional Action Issues 86
3.4.2 Action-independent Tabs 87
Defining Action Properties 87
Using the **Clone new folder** option 90
Defining Do This Properties 90
Defining Notes Properties 91
Defining Simulation Properties 92
Defining a Creation Action's Simulation Properties 92
Defining Simulation Properties for Later Actions 93
3.4.3 Action-specific Properties 94
Defining a User Action's Form Properties 94
Defining a User Action's Roles Properties 96
Defining a Loopback Action's Roles Properties 96
Defining a Timed Action's Time Properties 98
Defining a Timed Action's Roles Properties 99
Defining a Flagged Action's Flag Properties 100
Defining a Flagged Action's Roles Properties 101
Defining a Rendezvous Action's Maps Properties 102
4 CUSTOM AND SYSTEM VARIABLES 103
4.1 Custom Variables 103
4.1.1 Custom Variables and Forms 104
4.1.2 Creating a Custom Variable 104
4.2 System Variables 106
4.3 Folder Variables 107

4.4 Assigning a Value 108

5 COMPLEX TYPES 110

5.1.1 Generating a Complex Types Library 110
5.1.2 Retrieving Data from Complex Types 111
5.1.3 Retrieving data into fields 111
5.1.4 Populating a List box 114
5.1.5 Populating a Dropdown 114

5.2 Map Complex Type Dialog 115

5.2.1 Columns 116
5.2.2 Toolbar 116
5.2.3 Return Value 117
5.2.4 Serialization 117
Serializing 118
Right Mouse Menu 118

5.3 ComplexTypesUtils Support Library 119

5.3.1 FormatComplexType 119
Parameters: 119
Returns: 119
FormatComplexType Example 119
5.3.2 FormatComplexTypeDirect 121
Parameters: 121
Returns: 122
FormatComplexType Example 122
Using FormatComplexTypeDirect 122
5.3.3 GetList 122
Parameters: 123
Returns: 123
GetList Example 123
5.3.4 GetListDirect 124
Parameters: 124
Returns: 125
5.3.5 GetNameValueList 125
Parameters: 125
Returns: 125
5.3.6 GetNameValueList Direct 125
Parameters: 125
Returns: 126

6 FORMS 127

6.1 The Form Editor 127

6.1.1 The Form Toolbar 128
6.1.2 Working with Form Elements 128
Adding Elements to a Form 129
Selecting Form Elements 129
Moving Form Elements 130
Copying and Pasting Form Elements 130
Aligning Form Elements 130
Determining Form Element Layering 130
Deleting Elements from a Form 131

Placing Form Segments on a Form 131
6.2 Properties editor 131
6.3 Creating a New Form 132
6.3.1 Defining Form Properties 133
Defining a Form's Form Properties 133
Defining a Form's Format Properties 135
Defining a Form's Do This Properties 137
Defining a Form's Notes Properties 138
6.4 Form Element Properties 138
6.4.1 Dependencies 138
6.4.2 Client Extensions 139
6.4.3 Defining Element-independent Properties 139
Defining a Form Element's Extra Properties 140
Defining a Form Element's Notes Properties 142
6.4.4 Defining Element-specific Properties 142
Defining a Frame or a Rule 142
Defining a Frame/Rule's Rule Properties 143
Defining an Image 143
Defining an Image's Image Properties 144
Defining an Image's Do This Properties 145
Defining a Label 145
Defining a Label's Label Properties 146
Defining a Label's Do This Properties 147
Using a Label field to add a link to the form 147
Using a Label field to add HTML elements to the form 148
Defining an Attachment Clip 149
Defining an Attachment Clip's Do This Properties 151
Defining a Command Button 152
Defining a Command Button's Button Properties 153
Defining a Command Button's Do This Properties 154
Defining a Check Field 155
Defining a Check Field's Check Properties 156
Defining a Check Field's Do This Properties 157
Defining a Currency Field 158
Defining a Currency Field's Currency Properties 159
Defining a Currency Field's Options Properties 160
Defining a Currency Field's Do this Properties 161
Defining a Dataset Field 162
Defining a Dataset Field's Dataset Properties 163
Defining a Dataset Field's Do This Properties 165
Defining a Date/Time Field 166
Defining a Date/Time Field's Date/Time Properties 166
Defining a Date/Time Field's Options Properties 168
Defining a Date/Time Field's Do This Properties 169
Defining a Drop-down Field 170
Defining a Drop-down Field's Drop-down Properties 171
Defining a Drop-down Field's Options Properties 173
Defining a Drop-down Field's Do This Properties 177
Defining a Default Value for a Drop-down Field 178
Defining a List Field 179
Defining a List Field's List Properties 179
Defining a List Field's Options Properties 181
Defining a List Field's Do This Properties 182

Defining a Memo Field ... 183
Defining a Memo Field's Memo Properties ... 184
Defining a Memo Field's Do This Properties ... 185
Defining a Number Field ... 186
Defining a Number Field's Number Properties ... 187
Defining a Number Field's Options Properties ... 189
Defining a Number Field's Do This Properties ... 190
Defining a Radio Group Field ... 191
Defining a Radio Group Field's Radio Group Properties ... 192
Defining a Radio Group Field's Options Properties ... 193
Defining a Radio Group Field's Do This Properties ... 194
Defining a Signature Field ... 195
Defining a Signature Field's Signature Properties ... 196
Defining a Signature Field's Do this Properties ... 197
Defining a Status Field ... 198
Example One ... 199
Example Two ... 200
Example Three ... 201
Defining a Status Field's Status Properties ... 202
Defining a Status Field's Options Properties ... 203
Defining a Status Field's Do this Properties ... 204
Defining a Text Field ... 204
Defining a Text Field's Text Properties ... 205
Defining a Text Field's Options Properties ... 206
Defining a Text Field's Do this Properties ... 209
Using a Text field to add a link to the form ... 210
Defining a Grid ... 212
Grid Types ... 212
Defining a Grid's Grid Properties ... 214
Defining a Grid's Options Properties ... 215
Defining a Grid's Columns Properties ... 216
Defining a Grid's Do This Properties ... 220
Using a Grid to set up Folder-on-Folder ... 221
Avoiding Common Editable Grid Errors ... 223

6.5 External Forms ... 224

6.5.1 Defining External Forms Options ... 224
To Add, Modify, or Review External Forms Options ... 224

6.5.2 Defining External Form Properties ... 226
Defining an External Form's Form Properties ... 227
Defining an External Form's Format Properties ... 229
Defining an External Form's Do This Properties ... 230
Defining an External Form's Notes Properties ... 231

6.5.3 External Forms Issues ... 231
Euro symbols in PDF forms ... 231
Viewing PDF files ... 231
Integration Wizard ... 231

7 DOCUMENT MANAGEMENT SUPPORT ... 232

7.1 What does Metastorm DMS provide? ... 232

7.2 Setup ... 233

7.2.1 Procedure Properties ... 233

7.3 DMS Attachment Clips ... 234

7.3.1 DMS Single Attachment Clip ... 234

 © Metastorm Inc., 2008

7.3.2 DMS Multiple Attachment Clip *235*
7.3.3 DMS Properties *236*
7.3.4 Dependants *237*
7.3.5 Edit Only *237*
7.3.6 Creating a Dynamic Default Location *237*
7.4 Integration Wizard **237**
7.4.1 Common DMS Integration Wizard Commands *238*
Get Document Id From Clip 238
Get Selected Document Id 238
Get Folder Document Ids 238
7.4.2 SharePoint Integration Wizard Commands *238*
Upload Document 238
Browse SharePoint 239
Check In Document 239
Check Out Document 239
Delete Document 239
Download Document 240
Get Document Meta Data 240
Set Document Meta Data 240
Undo Check Out Document 241
Verify Document 241
Get Document Author 241
Get Document Checked Out By 241
Get Document Creation Date 242
Get Document Editor 242
Get Document Last Modified 242
Get Document Title 242
Verify Folder 242
8 DEFINING ROLES **244**
8.1 Defining a Role **244**
8.1.1 Identifying the Roles in Your Organization *246*
8.1.2 User Attributes and Relationships *246*
Attributes 246
Relationships 247
8.1.3 Defining Roles Using Metastorm Data Extraction Utilities *248*
8.1.4 Adding Roles to the Roles List *248*
8.1.5 Deleting Roles from the Roles List *250*
8.1.6 Editing Roles *250*
8.1.7 Defining Dynamic Role Formulas *251*
Using operators within dynamic role formulas 251
Example One 252
Example Two 252
Using %LDAPSearch within dynamic role formulas 252
8.1.8 Using the 'everybody' role *252*
Workaround 253
9 DEFINING FLAGS **254**
9.1 Flags **254**
9.2 Defining a Flag **255**
9.2.1 Flag Data *255*

9.2.2 Flags Dialog *255*
9.2.3 Flags Dialog Buttons *256*
9.3 Raising Flags from within Procedures **257**
9.3.1 Properties editor *257*
9.3.2 Using eraiseflag.exe to raise a flag from an external application *258*
9.3.3 Receiving Flag Data *259*
10 CREATING FORMULAS **261**
10.1 Integration Wizard **261**
10.1.1 The Integration Wizard Interface *263*
10.1.2 Using the Integration Wizard to Create a New Formula *264*
Integration Wizard Example: Assign Value 265
Integration Wizard Example: Conditional Operation 269
10.1.3 Condition Builder *270*
10.1.4 Add-ins *271*
10.1.5 Formula Editor *271*
Using the Formula Editor 271
10.1.6 Accessing Databases *272*
Table Name(s) Argument 273
Column Name(s) Argument 273
Column Values Argument 274
Data Source Argument 274
User Name Argument 274
Password Argument 274
Row Condition(s) Argument 274
10.1.7 Accessing Directories (Direct Directory Access) *274*
Setting up the LDAP alias 274
Searching the Directory 276
10.1.8 Custom Variables *278*
10.1.9 Defining External Tables *280*
Creating External Tables 280
10.1.10 Accessing Business Rules Engines *282*
Selecting the 'Execute a Rule' or 'Get Rule Result' Integration Wizard Item 284
Specifying Rule Connection Information 285
Selecting the Rule 287
Specifying Custom Variables to Pass to the Rule 287
10.2 Scripting **287**
10.2.1 Working with the Scripts dialog *289*
Adding a Script 290
10.2.2 Calling Server-side Scripts *292*
Calling a Server-side Script through the Integration wizard 292
Calling a Server-side Script through the Formula Editor 295
10.2.3 Calling Client-side Scripts *297*
Calling a Client-side Script through the Integration wizard 298
Calling a Client-side Script through the Formula Editor 301
Client Event Limitations 304
Field exit and entry 304
Client Side Script Functions 305
10.3 Glossary of Integration Wizard Items **305**

APPENDIX A - SAMPLE PROCEDURES 321

Publishing a Sample Procedure 321

Locating the sample procedures 321
Using the sample library 322
Setting roles for the sample procedures 322
Using the Set Up sample procedure 322
Sample Library 322
Points to Note 322
Set Up 323
Preparation 324
Points to Note 324
Case Management 325
Preparation 325
Points to Note 325
Flight 327
Preparation 327
Points to Note 327
Lead Tracking 329
Preparation 329
Points to Note 329
New Hire 331
Preparation 333
Points to Note 333
One Call 335
Preparation 335
Points to Note 335
Purchase Order 337
Preparation 337
Points to Note 337
Training 339
Preparation 340
Points to Note 340
Wizard 342
Preparation 342
Points to Note 342
BRI 343
Preparation 344
Points to Note 344
External PO 345
Points to Note 345
Send Mail 346
Points to Note 346
News 347

APPENDIX B - FORMULA COMPONENTS 348

Type-face Conventions 348
General Syntax 348
Formula Types 350
Type Conversions 350

Strings and Quotes **351**
Where Formulas May Be Used **352**
When Formulas Are Evaluated **353**
Evaluation of 'When…' Formulas **355**
Forcing Failure *356*
Common Stages *356*
System Variables **356**
%Action.Name *356*
%Action.Notes *357*
%Action.StartsStage *357*
%Form.Name *357*
%Procedure.Name *358*
%Procedure.Version *358*
%Session.FlagName *358*
%Session.FlagData[column_number] *358*
%Session.FlagRaiser *359*
%System.Name *359*
%System.Roles *359*
%System.Time *359*
%System.Users *359*
%User.Error *360*
%User.Form *360*
%User.Input *360*
%User.Input[column_number] *360*
%User.Name *361*
Folder Variables **361**
%ActionCount *361*
%Archived *361*
%Category *361*
%CreationTime *361*
%Deadline *362*
%EntryTime *362*
%FolderID *362*
%FolderName *362*
%MapName *362*
%Message[event_number] *362*
%Notes[event_number] *363*
%Originator *363*
%Parent *363*
%Priority *363*
%ServerName *363*
%StageName *364*
%Subject *364*
%Updated *364*
custom variables… *364*
Form Variables **364**

%Field.[fieldname] *364*

Pure Functions **365**

%Abs(value) *365*
%Asc(character) *365*
%Chr(code) *366*
%Concatenate(string { , string }) *366*
%Count(list [, delimiter]) *367*
% Duration (from, to [, units]) *367*
%EMPTY() *368*
%Find(substring, instring, case [, start_position]) *368*
%FindEntry(list, item [, delimiter]) *369*
%FormatCurrency(currency-value, [locale, [digits, leading, group, decimal-separator, thousand-separator, negative-order, positive-order, currency-symbol]]) *369*
%FormatNumber(number, decimals [, width [, format]]) *371*
%FormatTime(timestamp, format) *371*
%GetEntry(list, index [, delimiter]) *373*
%Length(string) *373*
%LINEFEED() *373*
%MakeDate(count, units, direction, from) *374*
%Max(value1 , [value2]) *375*
%Max(list) *375*
%Min(value1 , [value2]) *375*
%Min(list) *376*
%Months(start-date, months) *376*
%NEVER() *376*
%NEWLINE() *377*
%Rem(dividend, divisor) *377*
%RemoveEntry(list, index [, delimiter]) *377*
%Replace(source, old, new [, count]) *378*
%Round(number [, decimals [, round-down]]) *378*
%Substring(start, length, string) *379*
%Weekdays(start-date, days) *379*

Access Functions **380**

%CaptureAttachment (folderID, filename, attachmentData) *380*
%Dir(path, [Alt Delimiter]) *380*
%EmailAddress(user, [Alt Delimiter]) *381*
%GetData(data_source, table, rows, column, [col_delimiter], [row_delimiter]) *381*
%GetFolder(table, folder, variable, [Alt Delimiter]) *382*
%GetText(filename) *382*
%Manager(userID, role [, fallback_role]) *383*
%ReadAttachment (folderID, filename) *383*
%ScriptEval(language, file, procedure, map, expression [, args]) *384*
%SelectSQL(statement [, data-source [, col_delimiter [, row_delimiter [, headers]]]]) *385*
%Staff(userID, role , fallback-role, [Alt Delimiter]) *387*
%ToDoList(folderID, [Alt Delimiter]) *387*
%WatchList(folderID, [Alt Delimiter]) *387*

%LDAPSearch(directory_alias, filter_expression, attribute_name, [search_scope], [Alt Delimiter]) ... *388*

Impure Functions ... **388**

%Delete(filename) ... *389*
%DeleteAttachment(folderID, filename) ... *389*
%EMail(toList, ccList, subject, message ⟦ , attachment ⟦ , reply address ⟧ ⟧) ... *390*
%ExecProc(procedure, arguments) ... *390*
%ExecSQL(statement ⟦ , data_source ⟧) ... *391*
%Execute(procedure, arguments) ... *392*
%ExecuteExtensionEval (JScript.NET, %Procedure.Name, %MapName, "Metastorm.ProcessEvents.HandleExternalEvents.DelegateEvents" ,) ... *392*
%ExecuteRule(ruleConnection,customVariables) ... *393*
%GetAttachment(folderID, filename, path) ... *394*
%NewAttachment(folderID, filename, path) ... *394*
%Print(document, folderID, printer, filename) ... *395*
%RaiseFlag(flag, folderID { , flag_data }) ... *396*
%RegisterInQueue (symbolicName, system, eworkHost, queueSource, credentials, process, clientType, map, stage, action, params) ... *396*
%RegisterOutQueue (symbolicName, system, queueName, params) ... *397*
%Run(application, arguments) ... *398*
%SaveData(folderID, filename, [Alt Delimiter]) ... *398*
%SendMQMessage (symbolicName, message, priority, timeToLive, folderID) ... *399*
%SendMQMessageEx (type, queueName, message, priority, timeToLive, params) ... *399*
%Script(language, file, procedure, map, statement [, args]) ... *400*
%ScriptAsync(language, file, procedure, map, statement [, args]) ... *401*
%UnregisterInQueue (symbolicName) ... *402*
%UnregisterOutQueue (symbolicName) ... *402*
%WriteText(string, filename ⟦ , new-file ⟧ saveasUnicode) ... *403*
Send an e-mail message ... *404*

Operators ... **405**

General Operators ... *405*
Typecast Operators ... *405*
Logical Operators ... *406*
Arithmetic Operators ... *406*
Quotation marks ... *407*
Escape operator ... *409*
Line separator ... *409*

APPENDIX C - DESIGNING AN ACCESSIBLE PROCESS ... **410**

Section 508 Compliance ... **410**
Keyboard Navigation ... **410**
Internet Explorer Text Size ... **411**
Color Blindness ... **411**
JAWS for Windows Support ... **411**

Placing fields ... *411*
Form fields ... *411*

Unsupported fields *412*
Invalid Characters *412*
Configuring JAWS **412**
Accessing Metastorm BPM via JAWS **412**
Determining whether you are logged in to a service *413*
Determining the current list *413*
Logging in to a service *414*
Navigating to the submit and cancel buttons on a form *414*
APPENDIX D – BUSINESS RULES WEB SERVICE CONFIGURATION PARAMETERS 415
Connection Parameters **415**
URL – string *415*
Timeout – long *415*
Authentication Parameters **415**
AuthMethod – long *415*
AuthUser – string *416*
AuthDomain – string *416*
AuthPassword – string *416*
ProxyServer – string *416*
ProxyUser – string *416*
ProxyPassword – string *417*
BypassProxyOnLocal – boolean *417*
Advanced Properties **417**
AllowAutoRedirect – boolean *417*
KeepAlive – boolean *417*
UserAgent – string *418*
PreAuthenticate – boolean *418*
RequestEncoding – long *418*
APPENDIX E – MASKS FORMAT **419**

Metastorm BPM Release 7.6

Designer User Manual

1 GETTING STARTED

In a traditional business environment, information may be passed between individuals in the form of memos, files, notes, and voice-mail messages. Ideally, team members receive all the information they require to complete their portion of a business process in a timely manner. This method, however, often causes problems. Pages of a document may be lost when being transferred from one team member to another. Important addresses and telephone numbers may be accidentally tossed in the garbage or buried under stacks of paperwork. Work grinds to a standstill or escalates to crisis management.

Metastorm BPM uses the concept of an electronic folder in which all information relevant to a particular task is placed. These folders are routed electronically from user to user as team members complete assigned activities. All the information necessary is gathered into a single location and made available to the participants as the project reaches their respective desktops. The chance of losing important information is minimized, and processes move smoothly toward completion.

In this chapter, we will introduce you to *Metastorm BPM* and discuss the following topics:

- Components
- Design Basics
- The Metastorm BPM Designer Interface
- Publishing the Procedure
- Overview of Metastorm Administrative Tools: Users and Roles, System Administrator, Services Manager

1.1 Metastorm BPM Components

Metastorm BPM is composed of the following components, each of which has its own system requirements. Specific system requirements are listed in the *Metastorm BPM Installation Prerequisites Guide*.

- **Metastorm Process Engine**

  For all applications providing Metastorm Client functionality, the Metastorm Engine is the connection point to the Metastorm database. It performs the processing. It is also the integration point for external applications, such as email systems and other applications.

- **Metastorm BPM Designer**

  The Designer is used to create a Metastorm procedure. Through the Designer, you create the maps and forms and build the business rules that are used to automate your business processes.

- **Administrative Tools**

  The Administrative tools set included with Metastorm (which includes *Users and Roles*, *System Administrator*, and *Services Manager*) lets you define your users and their roles in your organization, migrate your Metastorm database to a new schema and perform other administrative functions.

- **Directory Integration Suite**

  Metastorm BPM includes a Directory Integration Suite. The tools in this suite allow you to extract user-related attribute data out of data stores such as:

  - LDAP directories
  - Novell® Directory Services (NDS®)
  - Directories stored in an RDBMS

  This data is cached in the Metastorm database and becomes available to Metastorm processes

- **Metastorm Client Facilities**

  Metastorm Client facilities can be provided through multiple options, including:

  - **Web Extensions**

    This is an application, which runs on a web server. The web extensions allow users to access the Metastorm processes from any machine that has a web browser and internet functionality.

    Supported servers and browsers are listed in the Metastorm BPM Installation Prerequisites Guide *and* Supported Environments *document.*

    No Metastorm software needs to be installed on the machines where a browser is to be used to provide Metastorm Client facilities.

  - **Metastorm Web Parts for SharePoint Server**

    Metastorm provides a Metastorm web part for Microsoft® SharePoint Server. This web part acts as an interface for users to access Metastorm processes.

  - **email Clients**

    These are applications that allow users to access the Metastorm database information through Microsoft Outlook®.

- **3rd Party Client Applications**

  It is possible to provide Metastorm Client facilities through custom client applications developed by Metastorm business partners and other third-party developers using the Metastorm Software Development Kit (SDK).

- **Database** (third-party product)

  The information collected through a Metastorm process is stored in a database that is not a part of the Metastorm software. Metastorm BPM supports the use of Oracle® or Microsoft SQL Server™ for production systems and Microsoft Data Engine (MSDE) for testing and development.

All of these elements may be installed on the same computer or on separate computers (depending on available resources and your specific system configuration), as long as the elements have access to the information in the database, as shown in the following table:

Option	*Function*
Process Engine	→ Database
Designer	→ Database
Administrative Tools	→ Database
email Clients	→ Process Engine → Database
Web Client	→ Web Server → Process Engine → Database

*Table **1**: Metastorm Component Database Access Requirements*

Applications providing Metastorm client facilities require access to the database through the Metastorm Process Engine. These components will not function if you bypass the Process Engine.

1.2 Design Basics

Before using Metastorm BPM for the first time, you should be familiar with the terminology used in this environment and have a basic understanding of the concepts of business process design.

1.2.1 Concepts and Vocabulary

Familiarize yourself with the following Metastorm terms before proceeding. Each of these terms will be discussed in greater detail later in this manual.

For further information refer to the Metastorm Concepts guide.

- **Procedure**

  Metastorm BPM views the information, activities, and instructions required to automate a business process as a *procedure*. In Metastorm BPM, the main component of a procedure is one or more maps. In addition to the maps, a procedure may contain forms, roles, flags, external tables, and scripts. All of these components are stored in a single procedure file. Each of these items is described briefly below.

  - **Map**

    When you design a Metastorm process, it is represented through one or more *maps* (diagrams or process models), each illustrating the various steps required to complete

 © Metastorm Inc., 2008

a business process (the lifecycle of a folder). Each instance of the business process is called a *folder*, and the steps are called *stages* and *actions*.

- **Folder**

  A new folder, with a unique, system-generated ID, is created each time a new instance of the process is initiated. In this way, the database can track the information particular to each instance of a business process.

  A folder contains one or more pages (forms) of information relating to that instance of the process. This information may come from a variety of sources, such as:

  - Input by a user onto a form;
  - Data extracted from a database (internal or external); and
  - A file generated by another application.

- **Stages**

  When a folder reaches a user's desktop, Metastorm BPM views the folder as having reached a *stage* in the procedure. A map may also contain various system stages that do not require human interaction to move the folder to the next stage.

- **Actions**

  Metastorm BPM defines an *action* as the step or activity necessary to modify data and move a folder from one stage to the next. Actions may include activities such as:

  - Filling out a form
  - Logging a telephone call
  - Reviewing an attached file
  - Approving or denying a request.

  It is also possible to have actions that do not require human intervention, such as:

  - Determining the routing for a folder based on information available in a folder or in a database
  - Raising or responding to a flag
  - Moving a folder after a timed event
  - Starting an external application.

  You can set properties and formulas in the Process Designer to accommodate a wide variety of possible actions.

- **Forms**

  Within Metastorm BPM, you may create *forms*. These forms are used to gather and display information necessary to a business process.

- **Roles**

  Participants in a process have roles assigned to them based on either their individual or group responsibilities. Assignments within a procedure are made based on these role designations.

- **Flags**

  Flags are used to start or continue parts of an automated business process. There are two important aspects to the concept of a flag in Metastorm BPM: the Flag itself and the Flagged Action invoked by the flag.

  You can consider the Flag to be an "announcement" and the Flagged Action to represent "what should be done" following the announcement.

  For example, flags may be used with flagged actions to:

  - Create a new folder and pass information to another map
  - Release a folder from a stage where it has been held while awaiting the completion of some other action or event.

- **Publishing a Procedure**

  When a procedure has been created, it must be published before a client can access its blank forms and take part in the process. When a procedure is published, its components are stored in tables in the database.

- **Formula**

  Formulas can be used to direct the action of procedure components. They are evaluated by the Process Engine.

Refer to Section 10 and Appendix B for further information on Metastorm formulas.

- **Administration Forms**

  Administration forms can be used by the user to carry out administrative processes. Administration forms:

  - Do not start a process
  - Are not available for use in maps
  - Cannot access custom variables
  - Cannot access system folder variables
  - Are not associated with any folder
  - Cannot be renamed.

  Administration forms are automatically associated with a creation action that leads to an Archive stage.

Refer to the Administration Guide for details of the sample Administration forms.

- **External Tables**

  External tables can be used to store configuration data, reference table data, or any other data related to the procedure.

- **Scripts**

  A process can contain VBScript or JScript files that can be run client-side, and VBScript, JScript or JScript .NET files that can be run server-side.

- **Libraries**

  Libraries can be created to store map segments, form segments, scripts and LDAP aliases ready for re-use. When a library is associated with a new procedure its contents become available to that procedure.

Form and map segments are described in more detail in section 2.

1.2.2 Designing a Procedure

As you design a Metastorm procedure, you may ask the following questions:

- What action initiates the process?
- What tasks must be performed to complete the process? In what order must these tasks be performed?
- Which individuals within the organization will be involved? What roles will they be taking in the process?
- What information needs to be gathered during the course of the process? What information should be shared among participants?
- What are the possible outcomes? Does the procedure flow in a step-by-step manner from start to finish, or are there alternative paths to be taken into consideration? Are actions taken sequentially or in parallel?

The answers to these questions will help you to develop your procedure.

Define the Procedure Properties

Metastorm BPM allows you to identify certain properties for each procedure you design. The **Properties** dialog is available from the **File** menu. There are four tabs on the Procedure Properties dialog: **General**, **Used Libraries**, **LDAP Aliases** and **Document Management**.

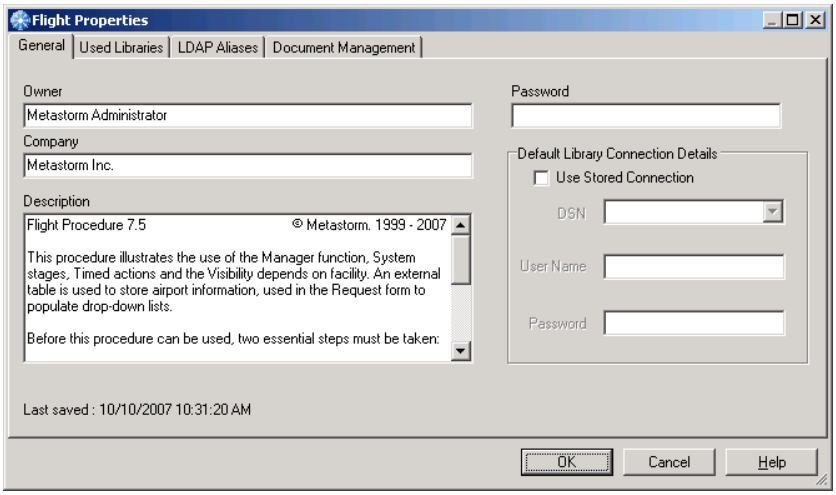


Figure 1: Procedure Properties Dialog

The **General** tab, shown in *Figure 1* above; allows input of:

- The name of the **Owner** of the procedure.
- The **Company** that owns the procedure.
- A **Description** of the procedure.
- An optional **Password** to protect the opening of the procedure file.
- The **Default database connection details**, used to load libraries for use by the current procedure.

If the password is lost, then the file cannot be opened. However, the file could be recreated by retrieving the procedure from the server.

In addition, the Properties dialogue also displays the dates and times the procedure was last published and saved.

The **Used Libraries** tab displays a list of the components contained within each library associated with the procedure and their last updated dates, see *Figure 2* below. When a library on the list is selected, it may be deleted or updated using the **Delete** or **Update** button. The **Add** button allows you to retrieve a library and associate it with the procedure.

All libraries must be published before they can be used within a procedure.

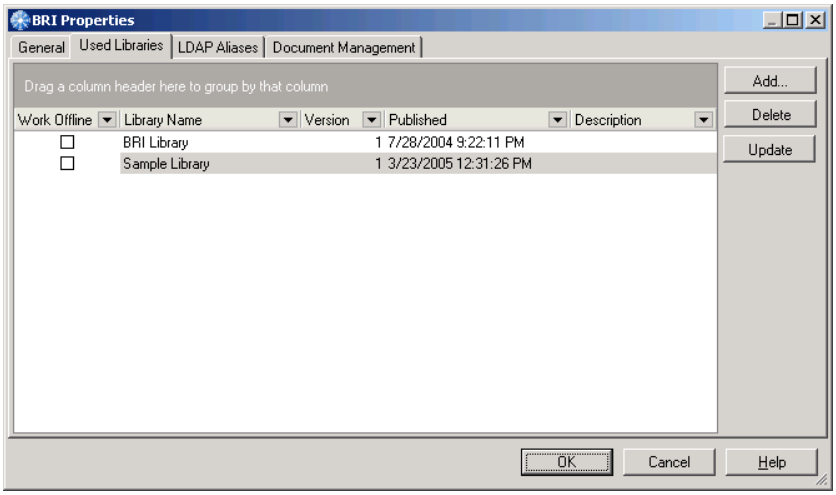


Figure 2: Used Libraries Tab

The **LDAP Aliases** tab displays a list of the LDAP Aliases associated with each library currently used by the procedure, see *Figure 3* below. The LDAP Aliases are created within each library and are available to the procedure when the published library is in use.

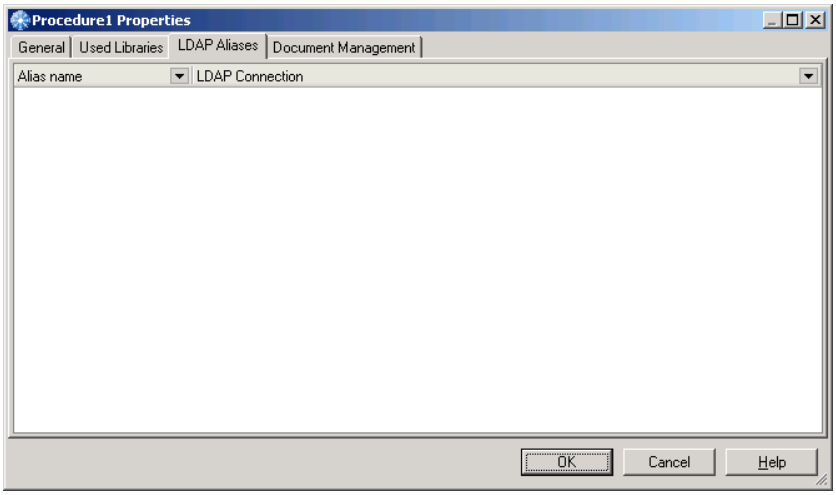


Figure 3: LDAP Aliases Tab

The Document Management tab displays the name of the DMS Provider and the default location of the DMS Provider.

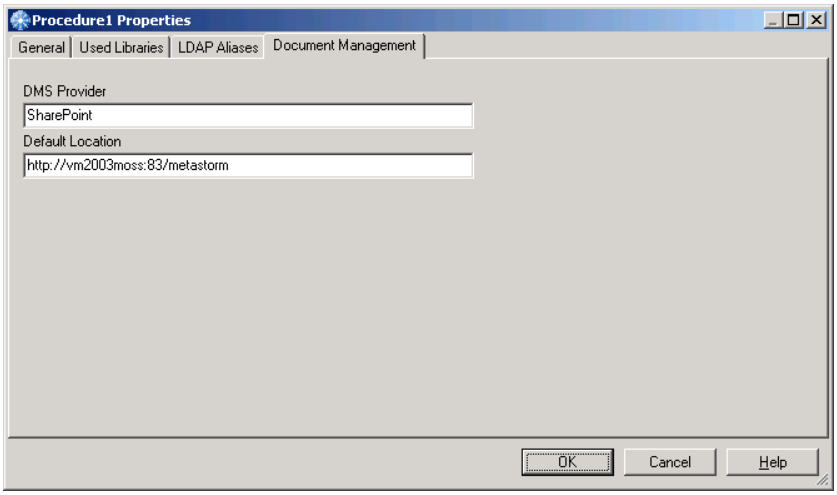


Figure 4: Document Management tab

For further information, refer to section 7 Document Management Support.

Define the Initial Action

When you define a new procedure, Metastorm BPM automatically creates a Start Box on the procedure (represented by a "clapperboard" icon). One or more initial user actions may be set to originate from the "clapperboard" as input into any stage. Each action must have a unique name and may be associated with a different form. Examples of such initial actions would be an incoming phone call or email message, a request for information or services, the beginning of multiple new projects, etc.

Define and Prioritize Tasks

The tasks required to complete the business process will be represented in your procedure as *stages* and *actions*:

- Stages act as resting points within a business process. At these stages, the Process Engine or designated users can act on the folders.
- Actions represent the tasks to be performed at various stages throughout the procedure. Actions may be:
  - Complex – For example, they may require the participant to enter information on a form.
  - Simple – For example, they may require approval before being passed on to the next stage of the procedure.

  Other actions may be performed by the system based on information pulled from the database or provided by a user at an earlier point in the procedure.

Identify Participants

The participants in your procedure are identified by their *roles* within the procedure.

Keep in mind that the system may also be a participant in the procedure, handling automated tasks. In this case, the system will be represented by a system stage rather than by a role assignment.

Determine Necessary Information

The type of task being performed determines the information needed and the routing of that information. For example, if you are building a map to handle incoming support calls, you will need to collect information concerning:

- The customer (name, company, account number, contacts).
- The product identification (item, model number, purchase date, service agreement).
- The support call (details, action taken, follow-up calls, support representative handling the call).

The information you require will determine the *forms* you will need to create to accompany your procedure.

Determine Possible Outcomes

A procedure may have several possible outcomes that may affect the route a business process follows from start to finish. For example, in the sample Flight procedure (Designer/Sample Procedures/Flight) shown in *Figure 5* below:

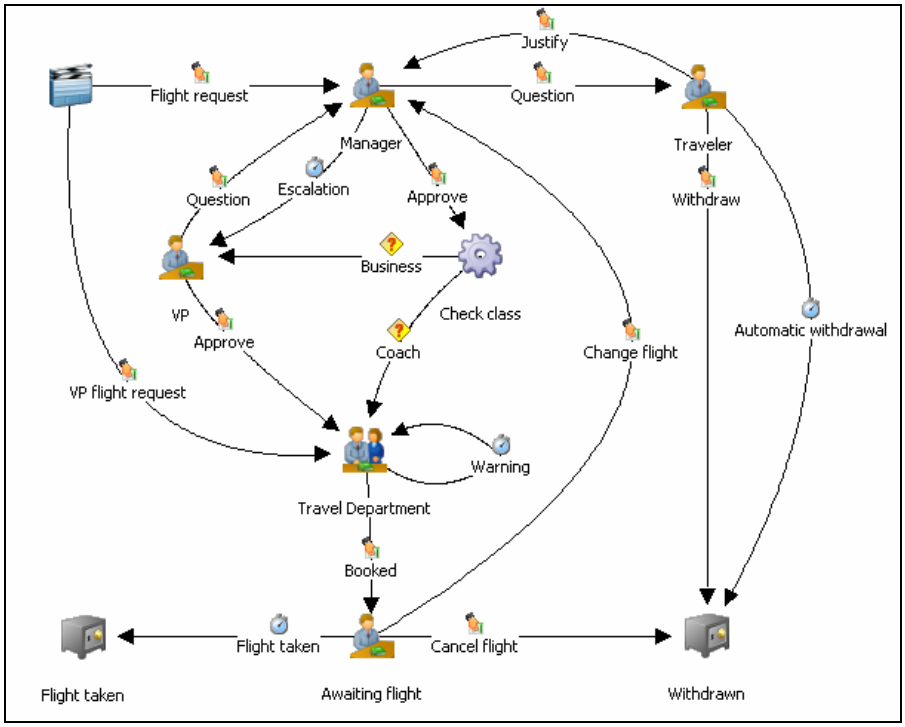


Figure 5: Flight Map

The map includes routings based on the following possible outcomes:

- An employee may be asked to justify their travel request.
- An employee may cancel or withdraw their travel request, thereby ending the procedure.

 © Metastorm Inc., 2008

- A request for a business-class flight is directed to a VP for second approval, while a request for a coach flight goes directly to the travel department after initial approval.
- Travel plans may be changed or cancelled, ending the procedure before the scheduled flight date.
- A request may be approved, travel scheduled, and the flight taken, ending the procedure.

Identify Reusable Elements

If you identify tasks that will be performed more than once within the process, consider creating a library. Libraries can be used to store common map segments, forms, form segments, scripts, roles, LDAP aliases and IW collections. When an existing library is associated with a procedure its contents become available to that procedure.

Refer to section 2 for information on using Metastorm libraries.

Run a Simulation of the Procedure and Analyze the Data

Metastorm BPM provides business managers with a modeling tool, which they may use to create Metastorm process maps, and run a simulation of the automated business process. The simulation data can then be analyzed through a separate simulation tool. For advice on use of the simulation tool contact Metastorm Inc.

By providing simulation data in the procedure and then exporting it to a simulation tool, business managers or process designers can test and refine the procedure during the design process.

Defining Simulation Data

The Properties editor displayed for actions on a Metastorm map includes a **Simulation** tab. This tab allows you to define the settings required to run a simulation of the procedure and process the results through a simulation tool. These settings include:

- The frequency rate for new folder creation
- The probability that a given action will progress the folder at a specified rate

Detailed instructions for defining simulation data can be found in the "Working with Actions" discussion in Section 3: Maps.

Maintain the Procedure

The **Find and Replace** feature allows the process designer to locate (and update) all the places that a given element is used in a procedure. This is particularly useful in a large procedure.

It is possible to find and replace the following element types:

- Custom variable
- Flag
- Role
- Form
- Field
- Form segment

- Client side script function
- Server side script function
- Database DSN
- Text (Sub Strings), and
- Test (Whole words).

To access the Find and Replace feature:

1. Select **Find and Replace** on the **Edit** menu or click on the Find and Replace icon on the General Toolbar, ;
    - The Find and Replace dialog is displayed:

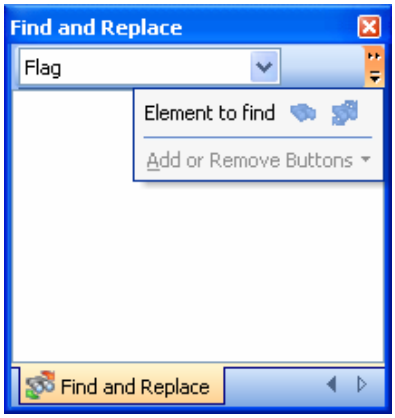


Figure 6: Find and Replace dialog

2. Select an element type from the Element type dropdown.
    - The **Element to find** dropdown is populated with the elements listed above.
3. Select or type an element in the **Element to find** dropdown.
4. Click on the ellipse in the top right of the Find and Replace dialog;
    - Find and Replace icons are displayed.

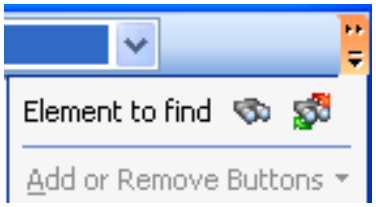


Figure 7: Find and Replace icons

5. Click on the **Find** button.
    - A tree representation of the relevant parts of the procedure is displayed in the **Occurrences found** box. This tree works in the same way as the Procedure Explorer. Double-click on an item to display it in the Designer's main pane.

For a description of the Procedure Explorer, refer to Section 1.3.4.

 © Metastorm Inc., 2008

6. To replace all occurrences of an element select the element type and element name, as described earlier in this subsection.
7. Click on the **Replace** button. A Rename dialog appears:

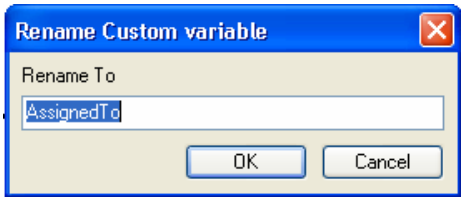


Figure 8: Rename dialog

8. Enter the new element name and click on the **OK** button.
    - The name of the element is replaced.

1.3 The Designer Interface

When you first start Metastorm Process Designer, a main window similar to the following is displayed:

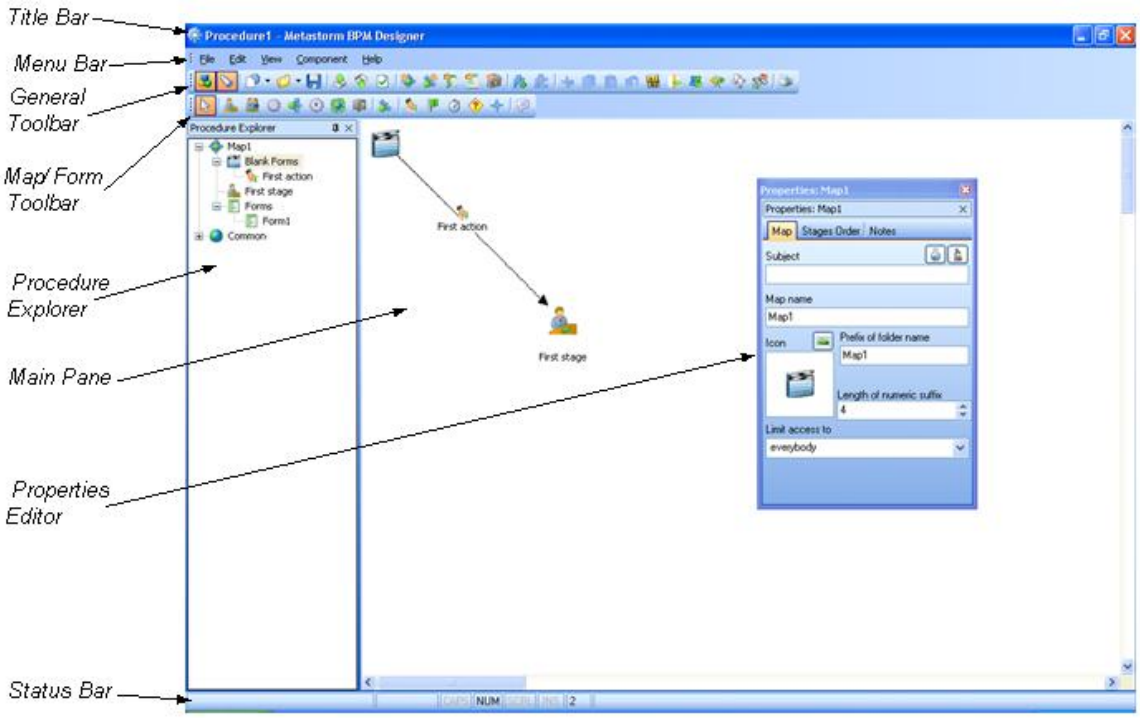


Figure 9: Metastorm Designer Main Window

The Designer main window has the following components:

- Title Bar
- Menu Bar
- General toolbar

- Map or Form toolbar
- Procedure Explorer
- Main Pane
- Properties Editor
- Status Bar

The display will vary according to the status of the Designer when it was last closed. When Designer is opened for the first time it will display only the initial process map.

1.3.1 Title Bar

The title bar displays the name of the current procedure.

1.3.2 Menus

You can gain access to most of the Process Designer's features by selecting items from either the menu bar or the buttons on the toolbar.

There are five menus that you can access from the menu bar:

- File.
- Edit.
- View.
- Component.
- Help.

The File Menu

You can access the File menu using the shortcut <**Alt**><**F**>. Shortcut and Access keys for accessing individual *File* menu options are given in *Table 2* below:

Option	*Access Key and Shortcut*	*Function*
New…	<Alt><F>+<N> or <Ctrl><N>	Starts a new procedure or library. If there is another procedure open, Designer gives you the option of saving changes to the current procedure.
Open…	<Alt><F>+<O> or <Ctrl><O>	Displays the standard File Open dialog, allowing you to select and open a previously saved procedure, with the option of first saving changes to the current procedure. This option also provides a list of the most recently opened procedures.
Save Procedure	<Alt><F>+<S> or <Ctrl><S>	Saves the open procedure.
Save As…	<Alt><F>+<A>	Copies the open procedure and displays the standard File Save dialog, allowing you to specify a new file name and location.
Save Definition…	<Alt><F>+<D>	Saves procedure details to a text file. Includes information about components and elements.

 © Metastorm Inc., 2008

Option	Access Key and Shortcut	Function
Close	<Alt><F>+<C>	Closes the procedure, with the option of first saving changes to the procedure.
Procedure Properties…	<Alt><F>+<E>	Displays the Procedure Properties dialog, which allows you to enter information pertaining to the Owner, Company, Password and Description associated with the procedure. The dialog will also display the date and time the procedure was last saved and published.
Publish…	<Alt><F>+<L>	Publishes the current procedure to the database. Only published procedures are accessible by registered Metastorm users.
Retrieve…	<Alt><F>+<V>	Displays the Retrieve Procedure dialog which allows you to select a previously published procedure and retrieve it from the database.
Validation	<Alt><F>+<T>	Checks the procedure for warnings or errors and displays them on the Validate Procedure dialog.
Print Definition…	<Alt><F>+<P>	Prints a full text definition of the current procedure, including all its components.
Print Component…	<Alt><F>+<R>	Prints the component currently displayed in the Main Pane.
Print Setup…	<Alt><F>+<U>	Reviews and allows changes to the print setup.
Exit	<Alt><F>+<X>	Closes Designer, with the option of saving changes to the current procedure.

Table 2: File Menu Options

The Edit Menu

You can access the Edit menu using the shortcut **<Alt><E>**. Shortcut and Access keys to the individual *Edit* menu options are given in the following table:

Option	Access Key and Shortcut	Function
Undo	<Alt><E>+<U> *or* <Ctrl><Z>	Reverses the last action.
Cut	<Alt><E>+<T> *or* <Ctrl><X>	Removes the items you have selected from the map or form. Designer places the cut items in your paste buffer. Cutting a stage from a map removes *all* actions linked to that stage.
Copy	<Alt><E>+<C> *or* <Ctrl><C>	Places a copy of the selected item in your paste buffer.
Paste	<Alt><E>+<P> *or* <Ctrl><V>	Inserts an item from the paste buffer into the map or form.
Delete	<Alt><E>+<D> *or* <Del>	Deletes the currently selected item or items from the map or form.

Option	Access Key and Shortcut	Function
Find and Replace…		Opens the Find and Replace dialog that allows you to search for a designated element type and rename it, and all references to it, as desired.
Select All	<Alt><E>+<L> *or* <Ctrl><A>	Selects all the items in the current map or form.
Move	<Alt><E>+<M>	Changes the source and destination stage for an action. It is hidden unless a map component is selected, and disabled unless an action is selected.
Align	<Alt><E>+<A>	Aligns the left, horizontal center, right, top, vertical center, or bottom of a group of elements.
Bring to Front	<Alt><E>+<F>	Used to specify which of a group of overlapping elements should be displayed in the foreground.
Send to Back	<Alt><E>+<B>	Used to specify which of a group of overlapping elements should be placed in the background.

Table 3: Edit Menu Options

The View Menu

You can access the View menu using the shortcut **<Alt><V>**. Access keys for the individual *View* menu options are given in the following table:

Option	Access key	Function
Properties	<Alt><V>+<P>	Toggles the Properties editor on/off.
Procedure Explorer	<Alt><V>+<B>	Toggles the Procedure Explorer on/off.
Custom Variables…	<Alt><V>+<V>	Displays a dialog that lists the custom variables defined for each map in the procedure, and lets you add or delete them.
External Tables…	<Alt><V>+<T>	Displays a dialog that lists the database tables defined in the procedure, and lets you add, delete, or rename them. Also allows you to add or edit columns for tables you have defined, or delete column definitions within the Designer for previously defined columns.
Roles	<Alt><V>+<R>	Displays the Roles dialog. Allows you to add, edit, or delete roles for the current procedure.
Flags	<Alt><V>+<F>	Displays the Flags dialog. Allows you to add, edit, or delete flags for the current procedure.
Scripts	<Alt><V>+<S>	Displays the Scripts dialog. Allows you to add, edit, rename, or delete scripts for the components of the current procedure.
Customize…	<Alt><V>+<C>	Displays a dialog containing a variety of options for customizing menus and toolbars.

 © Metastorm Inc., 2008

Option	Access key	Function
Options…		Opens the following tabs: Publisher - Allows for selection of publishing options; Editor - Font selection in the Script or Formula editor; Miscellaneous - confirmation on cancellation of Integration wizard dialogs and set up of component bar styles; File Location – Defines procedure, library and add-in file locations; and External forms – Defines the properties of forms to be imported.

Table 4: View Menu Options

The Component Menu

You can access this menu using the shortcut **<Alt><C>**. Access keys for individual *Component* menu options are given in *Table 5* below:

Option	Access key	Function
New …	<Alt><C>+<M>	Adds a new map, map segment, form, form segment, or administration form to the procedure.
Add…	<Alt><C>+<A>	Displays the standard *File Open* dialog, allowing you to specify an existing map, form, map segment, or form segment file you wish to add.
Remove…	<Alt><C>+<V>	Removes the selected component from the procedure. You cannot remove the last remaining map from a procedure.
Save As…	<Alt><C>+<S>	Copies the selected component and saves it under the new filename you select.
Save as Picture…	<Alt><C>+<P>	Saves the image of the selected item to a .BMP or .JPG file
Print Component…	<Alt><C>+<R>	Displays the standard *Print* dialog, allowing you to print the selected procedure component.

Table 5: Component Menu Options

The Help Menu

You can access the Help menu using the shortcut **<Alt><H>**. Access keys for individual *Help* menu options are given in *Table 6* below.

Option	Access key	Function
Contents and Index	<Alt><H>+<C>	Displays the Designer Help.
Metastorm Web Site	<Alt><H>+<M>	Opens a browser window that takes you to the Metastorm website (http://www.metastorm.com).
About…	<Alt><H>+<A>	Displays the Designer “About” box with release details.

Table 6: Help Menu Options

The Tools Menu

If any Process Designer add-ins has been specified, a Tools menu becomes available, containing a menu option for each available add-in.

You can access the Tools menu using the shortcut **<Alt><T>**.

Pop-up Menus

The following table shows the pop-up menus that are displayed when the right mouse button is clicked:

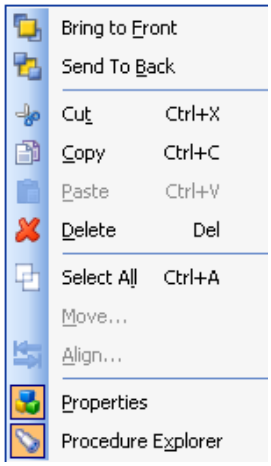
If the mouse is right-clicked …	*The following Pop-up Menu is displayed…*	*This Pop-up Menu contains items from the following menus:*
Anywhere in the Main Pane	 The Edit Menu The View Menu	The Edit menu The View menu
Anywhere in the Procedure Explorer		The Edit Menu The Component Menu

Table 7: Pop-up Menus

The active items displayed on the pop-up menus will vary based on the procedure components selected, or other actions performed in the creation of the procedure.

Dockable Menus

The Menu bar can be displayed anywhere on your desktop as a floating palette—or docked to the top, bottom, or either side of the main pane.

- To move the menu bar from its docked location, position the cursor over the ridge at the left of the toolbar and click and drag it to the new location. If you have resized your Designer window to be too narrow for all of the menu items on the Menu bar to be displayed, two chevrons will appear at the end of the visible portion of the bar. Clicking on the arrow will display the remainder of the menu items.
- To reposition the menu when it is a floating palette, click anywhere in its title bar and drag it to the desired location.

 © Metastorm Inc., 2008

Personalized Menus

As an additional feature, Designer menus can be set to self-optimize to provide the user with a personalized menu. You can customize your menus by:

1. Select **View | Customize**; or place the cursor anywhere in the Menu/Toolbar and *right-click:*
    - Select the **Customize** option to display the following dialog:

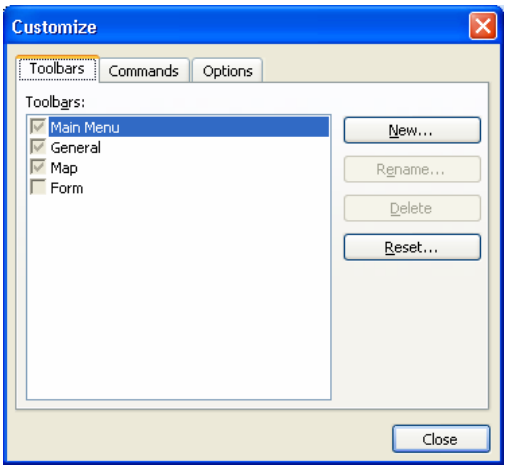


Figure 10: Customize Dialog

2. Check the **Main Menu** checkbox and select the **Options** tab;
    - The Options menu is displayed.

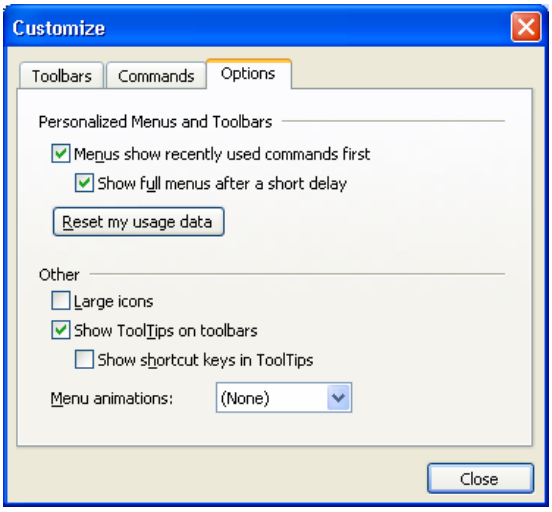


Figure 11: Menu Option Dialog

3. Make your selections from this dialog and select **Close**.

You may change your customization choices at any time.

The menu items described on the previous pages are presented in their default arrangement. If you customize your menus to show recently used commands first, the menu options described will not necessarily appear in the same order as described above.

1.3.3 Toolbars

There are three toolbars in the Designer main window:

- General toolbar;
- Map toolbar; and
- Form toolbar.

These toolbars provide access to the options you need to define, modify, and save procedures.

Each of these toolbars can be displayed anywhere on your screen as a floating palette, or docked to the top, bottom, or side of the main window.

- To move a toolbar from its docked location, position the cursor over the ridge at the left of the toolbar and click and drag it to the new location.
- To reposition a floating toolbar, click anywhere in its title bar and drag it to the desired location.

If you have resized the Designer to be too narrow for all of the buttons on the toolbars to be displayed, an arrow will appear at the end of the visible portion of the toolbar. Clicking on the arrow will display the remainder of the toolbar.

Toolbars can be customized by selecting **View | Customize** or right clicking anywhere on the Menu/toolbar area. The Customize toolbars dialog is displayed.

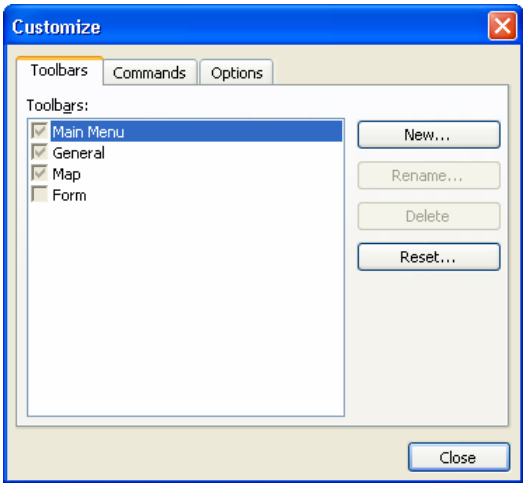


Figure 12: Customize Toolbars dialog

 © Metastorm Inc., 2008

General Toolbar

The General toolbar contains buttons generic to the whole application. You can use them to perform operations on the selected procedure component. The General toolbar contains the buttons shown in *Table 8* below:

Button	Function
Properties	Displays or hides the Properties editor window.
Procedure Explorer	Displays or hides the Procedure Explorer.
New	Starts a new procedure or library.
Open	Opens an existing procedure. Also provides a list of the most recently opened procedures.
Save Procedure	Saves the current procedure.
Publish	Saves and publishes the current procedure to the database.
Retrieve	Retrieves files from the database and opens them in Designer. The process designer may choose the version of the procedure that they wish to retrieve.
Validation	Displays or hides the Validation window. Validation checks a procedure and presents warnings and errors on the Validate Procedure dialog.
New Map	Creates a new map.
New Map Segment	Creates a new map segment.
New Form Segment	Creates a new form segment.
New Form	Creates a new form.
New Administration Form	Creates a new Administration Form.

Button	Function
Add Component	Adds an existing component from an external file to the procedure through a standard File Open dialog.
Remove Component	Removes the selected component from the procedure.
Cut	Removes the selected item from the map or form, placing it in the paste buffer.
Copy	Places a copy of the selected item in the paste buffer.
Paste	Inserts an item from the paste buffer into the map or form.
Undo	Reverses your last action.
Custom Variables	Opens a dialog that lists the custom variables for the current map, letting you add to or delete from the list.
External Tables	Displays a dialog that lists the external database tables defined in the procedure, and lets you add, delete, or rename them. Also allows you to add or edit columns for tables you have defined, or delete column definitions within the Designer for previously defined columns. External Table names cannot include extended ASCII characters. *Note: Not all databases support dropping a column. Therefore, Metastorm BPM does not remove the actual columns from external tables, even if references to those columns have been removed from the procedure.*
Roles	Displays the Roles dialog. Allows you to add, edit, or delete roles for the current procedure.
Flags	Displays the Flags dialog. Allows you to add, edit, or delete flags for the current procedure.
Scripts	Displays the *Scripts* dialog. Allows you to add, edit, rename, or delete scripts for the components of the current procedure.
Find and Replace	Displays the Find and Replace dialog that allows a search for a designated element type. Once located, the element and all references to it can be renamed.
Print Component	Prints the currently selected map or form.

Table 8: Buttons on the General toolbar

Map toolbar and Form toolbar

The Map toolbar and Form toolbar share the same default location, directly beneath the General toolbar; however, only one of these toolbars is available at a time. When a:

 © Metastorm Inc., 2008

- Map is selected in the Main Pane, the Map toolbar is displayed.
- Form is selected in the Main Pane, the Form toolbar is displayed.

You may choose move the Designer's toolbars to a more convenient location. However, regardless of their location, each will appear only when you are working with the corresponding procedure component.

You use the buttons on the Map and Form toolbars to create maps and forms in a procedure. The buttons available on the toolbars are identified in *Table 9* below. The functionality associated with these buttons will be described in greater detail in the sections on Map and Form creation:

Map Toolbar		***Form Toolbar***	
Button	***Button Name***	***Button***	***Button Name***
	Pointer arrow		Image
	User Stage		Status
	Group Stage		Label
	System Stage		Rule
	Sub-procedure Stage		Command Button
	Rules Stage		Attachment Clip
	Common Stage		Check
	Archive Stage		Number
	Use Map Segment		Currency
	User Action		Date/Time
	Flagged Action		Text
	Timed Action		Drop-down
	Conditional Action		Radio Group
	Rendezvous Action		List
	Comments		Memo
			Signature
			Grid
			Dataset
			Form Segment

Table 9: Buttons on the Map and Form toolbars

1.3.4 Procedure Explorer

When a procedure is open **Procedure Explorer** displays a tree representation of all its items. Procedure Explorer provides quick and easy navigation of the procedure. *Figure 13* below shows the Procedure Explorer when the Flight sample procedure is open.

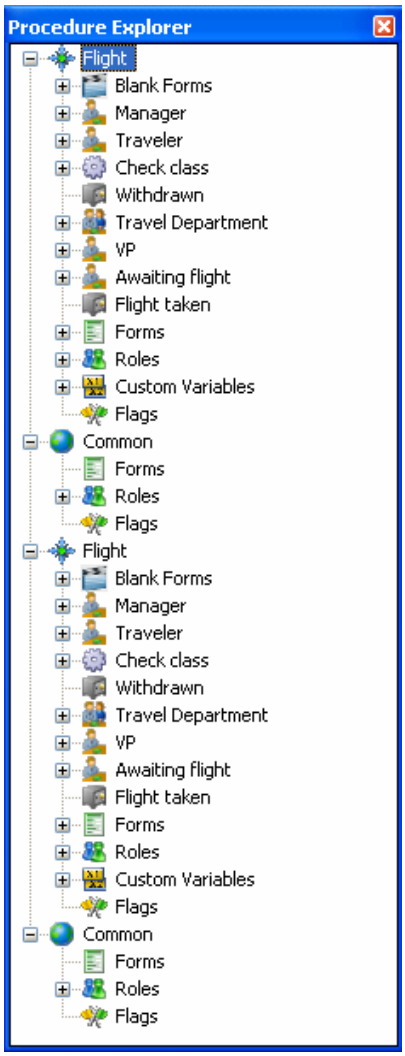


Figure 13: Procedure Explorer for the Flight procedure

The following table illustrates the items that are represented in the Procedure Explorer:

Parent Items	***Possible Child-Items***
Procedure (the whole tree)	Maps
	Map segments
	Administration forms
	Common elements
Map (and Map segment)	Scripts (server side and associated with a form)
	Blank Forms

 © Metastorm Inc., 2008

Parent Items	***Possible Child-Items***
	Stages
	Custom variables
	Forms
	Roles
	Flags
	Comments
Administration forms	Forms
	Custom variables
	Scripts (Server side and associated with a form)
Common elements	Forms
	Roles
	Flags
	Scripts (client and server side and common)
Blank Forms	(Creation) Actions
Stage	Actions
Form (and Form segment)	Form segments (sub element of form only)
	Controls
	Scripts (client side and associated with a form)
Comments	Comments

Table 10: Procedure Explorer elements

Procedure Explorer allows you to:

- View the details of an item. When you double-click on the item it is displayed or selected in the main pane.
- Drag and drop items where appropriate. For example it is possible to drag a form from one map to another.
- Select options via a pop-up menu.

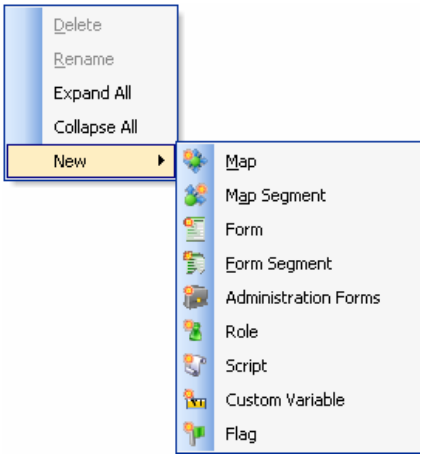


Figure 14: Procedure Explorer pop-up menu

From the pop-up menu it is possible to:

- Add new elements to the Procedure.

It is also possible to add new elements by clicking the appropriate button on the general toolbar.

- Rename the selected element.
- Add an existing map, form, map segment or form segment to the Procedure.
- Delete the selected element.
- Expand all nodes in the Procedure Explorer tree.
- Collapse all nodes in the Procedure Explorer tree.

1.3.5 Component Bar

The Component Bar provides an alternative to the Procedure Explorer to navigate procedures. To access the Component:

1. Select **View | Options**;
   - The Options Dialog is displayed:

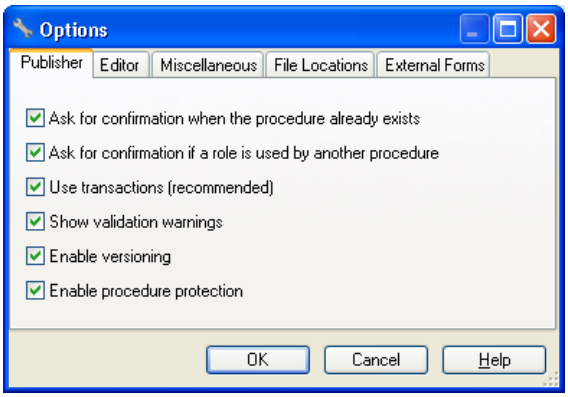


Figure 15: Options Dialog

2. In the Options Dialog click the **Miscellaneous** tab:
   - The Options display dialog is displayed:

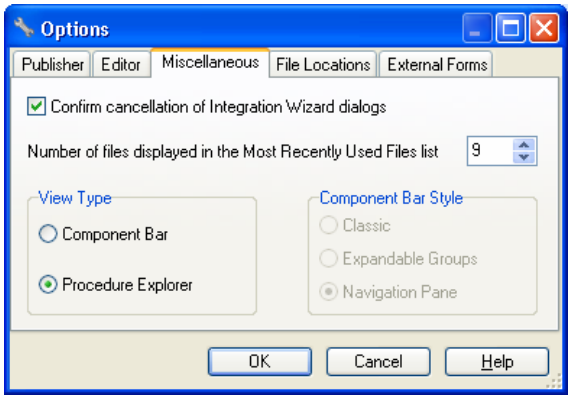


Figure 16: Options selection

 © Metastorm Inc., 2008

3. Ensure the **View Type** is set to **Component Bar**.
    - There are three formats for displaying the Component Bar.
    - *Figure 17* below shows the C**omponent Bar** in *Classic* format.

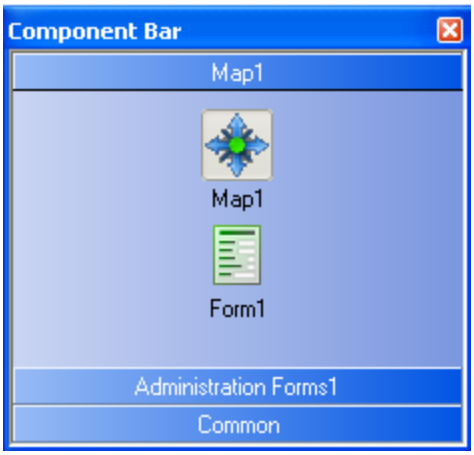


Figure 17: Component Bar – Classic style

When the Component Bar has been selected the Procedure Explorer icon operates on the Component Bar to hide or open the display.

When the Designer is first opened, the View Type is set to the Procedure Explorer. If the View Type is updated, this setting becomes the default setting.

The Component Bar can be used to add and delete procedure components, as well as to select the component you want displayed in the Main Pane. It has three types of display panes:

- **Map Group** (given the same name as the map)

  Each map in the procedure is identified by a tile in the Component Bar that bears the name of the map or map segment.

  Each map and map segment in the procedure is given an auto-generated name (i.e., Map1, Map2, Map3, etc.). Map and map segment names may be customized through the Properties editor or by right-clicking on the Component Bar and selecting the Rename Group option.

  Selecting a map group will open a display pane containing icons representing the map and each form associated with it. The list order can be modified by selecting an icon and dragging it to a new position in the list. Forms on one display pane can be relocated to another pane by dragging the corresponding icon up or down on the Component Bar onto the tile representing the desired new location. The pane will open, allowing you to place the form in the new pane. Maps may not be relocated.

  Custom variables used when defining a form are associated with a single map. If you relocate a form that uses one or more custom variables, it will not function properly until custom variables associated with that form are defined for the new map.

- **Administration Form Group**

  Each Administration form in the procedure is identified by a tile in the Component Bar that bears the name of the Administration Form.

- **Common Group**

  The Common display pane displays a separate icon for any forms not associated with a specific map. These forms may not access any custom variables, but are available for use with any map in the procedure. Fields on forms under the common tile may be associated with Metastorm system variables (such as Priority, FolderName, Originator, etc.) or may be calculated.

If your procedure contains several components and you wish to view more icons on the Component Bar, place your cursor at any point in the Component Bar, click on the right mouse button and select **Small Icons**.

1.3.6 Main Pane

This is where you define the elements to be used in the selected component of the current procedure. When you select a:

- Map or map segment on the Procedure Explorer, Designer displays the selected map in the Main Pane.
- Form or form segment on the Procedure Explorer, Designer displays the selected form in the Main Pane.

1.3.7 Status Bar

The Status Bar displays:

- The progress of any **File | Open** or **File | Save** operation
- The number of components in the current procedure
- Whether the procedure has been modified
- The status of the NUM lock and CAPS lock keys; and
- The status of the SCRL lock and INS lock keys.

Figure 18: Status Bar

1.3.8 Properties editor

Maps and forms and each of their elements have certain default properties; however, you may find it necessary to modify some or all of these properties. This is done using the Properties editor. Options available within the Properties editor vary based on the type of element selected.

 © Metastorm Inc., 2008

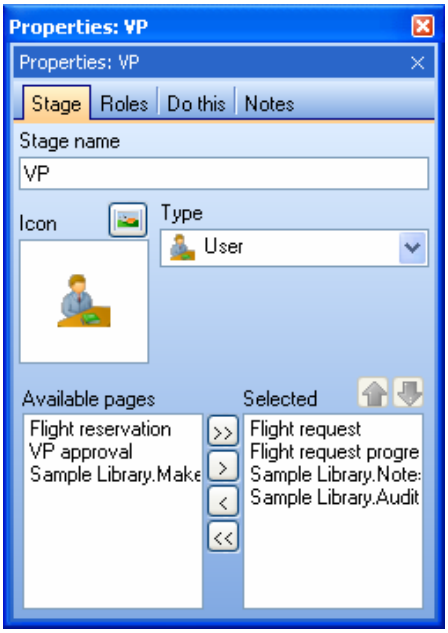


Figure 19: The Properties editor

The Properties editor can be considered as a master notebook of the modifiable properties of every component of your procedure (maps and forms, and the individual elements used to create them). Tabs within the Properties editor correspond to individual maps and forms, as well as each map and form element. The tabs of the Properties editor that you will see while creating your map depends on the type of map element selected and to the properties defined.

Each map or form element has a Properties editor tab specific to that type of element. Other tabs of the Properties editor are common to several different map and form elements, but may display different property fields or require instructions based on the type of map or form element you are working with. Throughout this User Manual, the individual tabs of the Properties editor will be described with the corresponding map or form element.

Many of the properties you can modify through the Properties editor require you to enter a formula. In most cases, the formula entry fields will include options for calling on the Integration wizard or Formula editor, both of which are described later in this section.

The Properties editor may be floating or docked to the top, bottom or sides of the main pane. To dock the Properties editor, position your mouse cursor anywhere in the title bar and drag it to the far right, far left or top or bottom of the main pane. Once the Properties editor has been docked it can be set to show and hide by selecting the auto-hide feature using *Push-Pin* on the Properties title bar.

Figure 20: Push -Pin to hide Properties Editor

When the auto-hide feature is selected the Properties editor will hide automatically when the cursor is moved out of focus. To make it show again, place the mouse cursor over the Properties tab that has appeared.

To undock the Properties editor when docked, position your mouse cursor over the Properties Editor's title bar and drag it to a new location within the main window.

1.4 Saving, Publishing, and Retrieving Procedures

The Designer provides methods for you to save your work in progress as well as options for returning to previous versions of a procedure as you refine your automated business process. Most of these options are available through the File menu and some may be accessed through General toolbar buttons.

1.4.1 File Locations

When you save or open a procedure or library, a standard **Save**, **Save As** or **Open** dialog is displayed. This allows you to locate the correct path or file.

It may be convenient to set the default file locations to the directories in which you store your procedures or libraries. To set the file locations:

1. Select **Options** from the **View** menu.
2. In the **Options** dialog, select the **File Locations** tab.
3. Place the cursor in the **Procedure Path Location** and click on the Browse buttons to set the default file locations for procedure and library files.
4. Click **OK** to save the settings.

If the File Locations are left blank, Process Designer will automatically open either the My Documents folder (the first time), or the last folder that was accessed from the dialog.

1.4.2 File Types

Metastorm Designer allows you to open and save procedures as .XEP files. These files store the procedure in XML format.

For details of the Metastorm process definition schema used in the XEP file format, refer to the Process Definition Schema Developer Guide available with the Metastorm Software Developer's Kit.

1.4.3 Saving a Procedure

As you develop a procedure, it is important to save your work in progress. Designer offers two options for saving a procedure: *Save Procedure* and *Save As*.

To save a procedure:

1. From the General toolbar, click on the **Save** button ; or
2. Select **File | Save Procedure** from the main menu.

Procedure names can use characters which are configured with a default locale of North American, Western European or Central European character sets. However, procedure names cannot include any of the following characters: Pipe (|), Period (.) Dollar Sign ($), Equal Sign (=), Back Slash (\), Quotation Mark (“), Forward Slash (/), Percent (%), Comma (,), Plus (+), Colon (:), Semi-colon (;) and Angle brackets (<>).

 © Metastorm Inc., 2008

To save the procedure under a different name, to a different location or as a different file type:

1. Select **File | Save As** from the main menu.
    - A standard **Save As** dialog will appear.
    - The procedure will be saved as an XEP file.

For information on file types, refer to section 1.4.2, File Types

2. Select a destination and file type, enter the filename, and click on **Save**.
    - Procedure and Library filenames may contain a maximum of 31 characters.
    - Designer will collect the current procedure elements and save the data.

1.4.4 Exporting a Procedure or Map

You can export a procedure to an alternative format if the relevant transformation instructions are available in the `Metastorm BPM\Designer` folder. The available formats will be listed in the **Save as type** dropdown on the **Save** dialog.

You can also export a map using the **Component | Save As** option.

Transformation instructions for exporting Metastorm maps to BPEL are provided with the Designer. To export a map as BPEL, select **BPEL files (*.BPEL)** in the **Save as type** dropdown when saving.

Metastorm BPM Designer can export to BPEL4WS 1.1. The later WS-BPEL 2.0 standard is not supported.

For information on producing transformations to allow export to alternative formats, refer to the Process Definition Schema Developer Guide available with the Metastorm Software Developer's Kit.

For information on the transformation that takes place when exporting to BPEL, refer to the BPEL Process Import and Export Guide available with the Metastorm Software Developer's Kit.

1.4.5 Opening a Procedure

To open an existing procedure file in Metastorm Designer for review or editing:

1. Click on the **Open** button ; or
2. Select **File | Open** from the main menu.
    - You can open procedures stored as XEP files.
    - Select the file type using the **Files of Type** dropdown.

For information on file types, refer to section 1.4.2, File Types

A procedure that has been saved, *but which has not yet been* published, *is not available for use by other users. It can be opened only in the Designer.*

1.4.6 Importing a Procedure or Map

You can import procedure data stored in an alternative format if the relevant transformation instructions are available in the `Metastorm BPM\Designer` folder. The available formats are listed in the **Files of type** dropdown on the Open dialog.

You can also import a map using the **Component | Add** option.

To import a BPEL procedure or map into Designer:

1. Select BPEL files (*.BPEL) in the Files of type dropdown.
2. Select the file to import.

Metastorm BPM Designer can import BPEL4WS 1.1. The later WS-BPEL 2.0 standard is not supported.

1.4.7 Validating a Procedure

After you have created a procedure, you should validate it to make sure there are no errors. Validation reviews the procedure to verify that no syntactical or design errors exist. The Validation function can be implemented when it is available on the Component Status toolbar.

To place the Validation function on the Component Status toolbar either click on the **Validate** button or select **File | Validate** from the menu.

To validate a procedure:

1. Open the Validation window by clicking on the **Validate** button.
    - The Validation dialog opens.
2. Click the Validate button at the top of the Validation dialog.
    - The procedure is validated and a report is recorded in the dialog pane.

1.4.8 Publishing a Procedure

Once all the components of a procedure have been defined, you must publish it to the database in order to make it available to authorized users of Metastorm BPM. Publishing a procedure creates the database entries necessary for the procedure to run in the Process Engine.

Using reserved words for the naming of process elements may prevent the published procedure from operating correctly. Metastorm BPM reserved words should not be used to name elements in a procedure.

To publish a procedure:

1. Click on the Publish button on the General toolbar ; or
2. From the menu select **File | Publish**.
    - A **Confirm** dialog states that the procedure will be saved if the Publish action is continued.

 © Metastorm Inc., 2008

3. Click on **Yes** to continue, or **No** to cancel.
    - If the procedure has not been saved previously, Designer displays the standard *Save As* dialog.
4. Specify a location and name for the procedure.

    It is not possible to publish procedures or libraries with the same name to the same database. However, if the database is case-sensitive, the same names in different cases are treated as different names. For example, in a case-sensitive database it is possible to publish the two procedures 'Flight' and 'FLIGHT'.

    - Designer displays the **Database connection details** dialog:

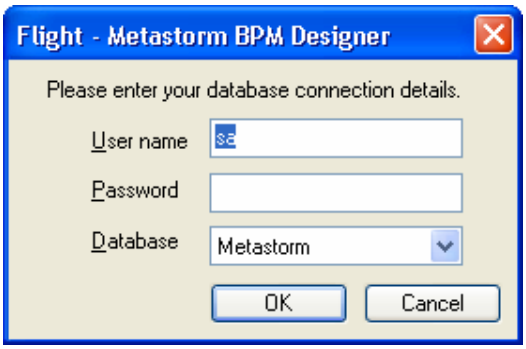


Figure 21: Database Connection Details Dialog Box

5. Enter the connection details for the database, and click **OK**.
    - The **Procedure** Protection dialog is displayed.

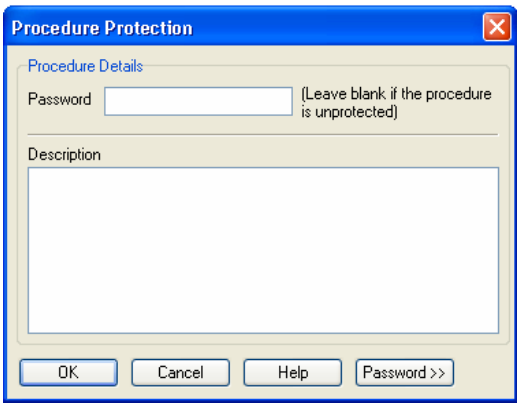


Figure 22: Procedure Protection dialog

    - Enter a **password** and a brief **description** of the procedure, if desired.
6. Click on **OK** to continue.
    - If you are updating or replacing a previous procedure of the same name, you will be asked to confirm your action.
7. Click on **Yes** to continue, or **No** to cancel.
    - Designer checks the procedure for syntax errors before publishing it. If there are no errors the procedure is published.

Metastorm BPM end- users may now access the published procedure in the system; however, before they may use the procedure, you must first assign roles to those users through the Users and Roles tool.

Data Formats

When a Procedure is published, data is stored in the database in Unicode format. All procedures and maps and the components that they contain are formulated as XML files, XEP or XEL, that contain characters defined and stored in Unicode format.

The particular Unicode format that is employed is determined by the database, it is not defined by the Designer. Designer determines the type of Unicode supported by the database by examining the data type in the eServerName column in the eServer table. The default value is Microsoft standard, UTF-16UCS-2.

If Designer determines that the database supports Unicode the progress dialog that appears when a procedure is published will announce that publication is in Unicode. *Figure 23* below shows a publishing dialog.

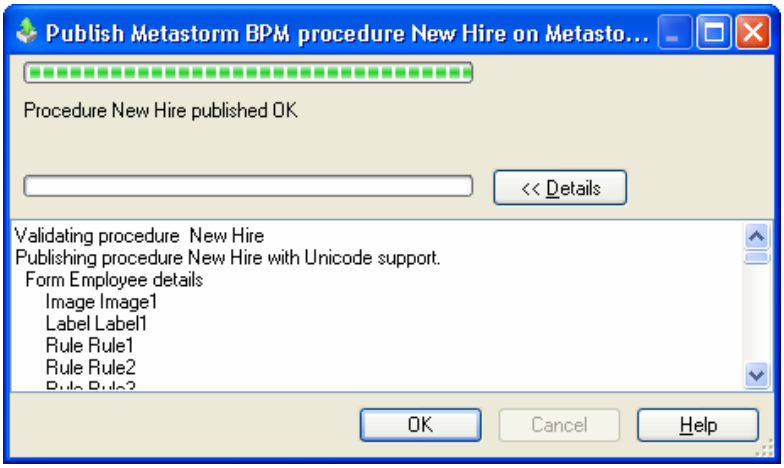


Figure 23: Announcement of Unicode support in Publishing

1.4.9 Retrieving a Published Procedure or Library

Each time you publish a procedure, Designer records version information in the database. This allows you to retrieve the most recent or previous versions of the procedure to refine the procedure or produce a template for a new business process.

Keep in mind that the active procedure available to your users is always the most recently published version.

To retrieve a procedure:

1. Click on the **Retrieve** button on the General toolbar ; or
2. Select File | Retrieve.
    - Metastorm BPM next displays the **Database connection details** dialog.

 © Metastorm Inc., 2008

3. Enter your information and press **OK** to continue.
    - Designer will display the **Retrieve** dialog:
    - Each published version of the procedure is listed, along with any descriptive information provided.

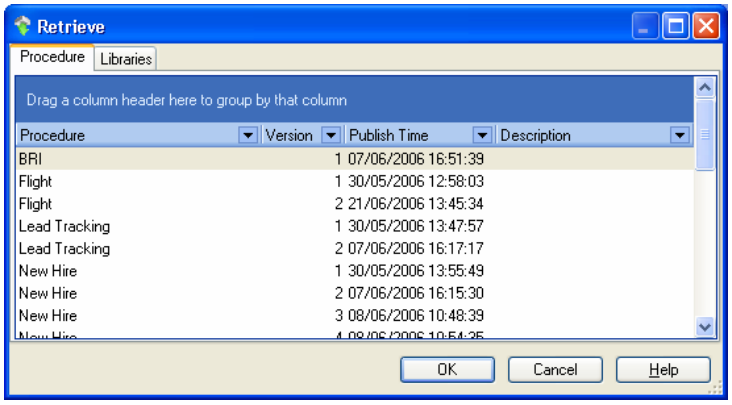


Figure 24: Retrieve dialog

4. Select the Procedure tab then select the procedure you wish to retrieve from the list.
5. Click on **OK** to continue.
6. To retrieve a library, select the Libraries tab then select the library you wish to retrieve from the list displayed.
    - Each published version of the library is listed, along with any descriptive information provided by the designer.

Click any column header to sort the list in ascending or descending order. To sort by multiple columns, hold down the Shift key when clicking the columns. To remove sorting, hold down the Control key and click the columns.

7. When you have made your selection, click on **OK** to continue.
    - The Designer will request a password for this procedure.
8. Enter the password and click OK to continue.
    - The Designer displays the **Retrieve Procedure to** dialog, which is the standard Save dialog.
9. Enter a filename and location for where the procedure will be retrieved.
    - The existing filename and last used directory are supplied by default.
10. To save this procedure file under a different name or to a different location, enter this information.

Click on **Save** to continue.

- The Designer will save the retrieved procedure to the location you specified, and open the procedure in the Designer.

11. Review or modify it as necessary.

2 LIBRARIES

A library can be used to store sub-sections of procedures that can be used in other procedures. Forms, form segments, map segments, scripts and LDAP aliases can be created once, stored in a library, and reused many times in procedures. When a library is associated with a procedure its contents become available to that procedure.

The benefits of using libraries include:

- New procedures may be created more quickly – existing libraries may be associated with new procedures.
- Maintenance of procedures is simpler – if a form that is used in several procedures needs to be updated, it need only be updated in the library. Once the procedure is republished with the latest version of the library the updated form becomes available to users of that procedure.
- Increased accuracy – by maintaining one copy of a commonly used part, the likelihood of introducing an error is minimized.
- More than one process designer may build procedures simultaneously – each process designer works on a library. Libraries are then brought together in one procedure file.

In addition, libraries may be taken offline. An offline library is one that is accessed from a file rather than from the database. This facility allows process designers without access to the Metastorm database to work on procedures.

2.1 Creating a Library

A library is a special type of file created in the Designer. The following may be included in a library:

- Forms
- Form segments
- Map segments
- Scripts

 © Metastorm Inc., 2008

- LDAP Aliases
- Integration Wizard collection.

This section describes how to create a new library, add to the library and save and publish the library.

2.1.1 Creating a new library

To create a new library:

1. Select File | New | Library.
    - A new empty library is displayed.
    - Any previously opened procedure or library is closed;
    - **Procedure Explorer** contains one **Common** item.
2. Select File | Library Properties;
    - The **Libraries Properties** dialog is displayed.

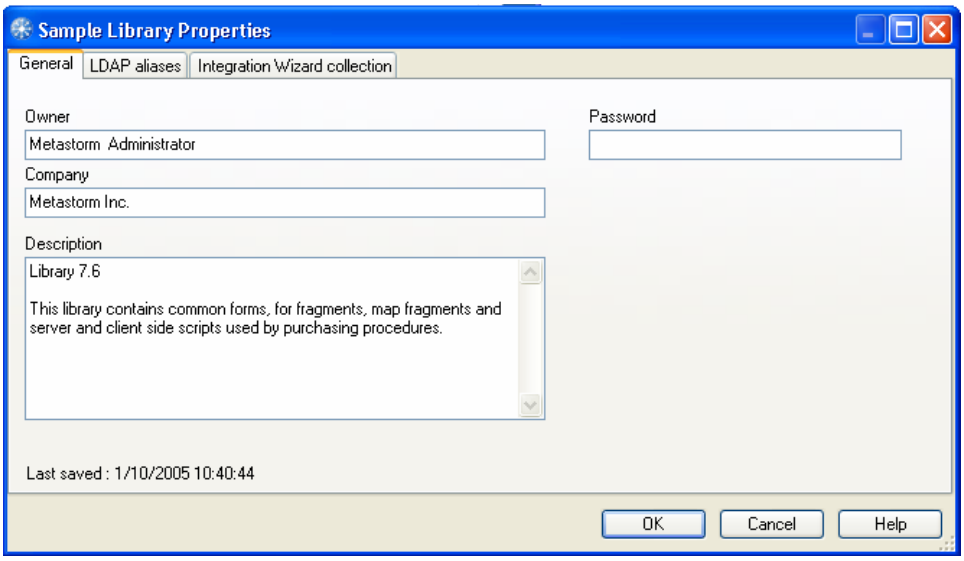


Figure 25: Library Properties

3. In the Libraries Properties the **General** tab allows enter:
    - The name of the owner of the library.
    - The organization that owns the library.
    - A description of the library.
    - An optional password to protect the opening of the library file.

    If the password were lost, then the file could not be opened. However, the file could be recreated by retrieving the library from the database.

In addition, the **Properties** dialog displays the dates and times the library was last published and saved.

The **LDAP aliases** tab contains a list of the LDAP aliases associated with the library.

The **Integration Wizard collection** tab allows you to add a file that extends the list of functions available in the Integration Wizard.

2.1.2 Adding library parts

This section gives an overview of the procedure components that can be stored in a library, and describes how to add each to a library.

Forms

In an organization a particular form may be used in more than one process. For example, the same audit trail form may be used in several procedures. The Metastorm sample library, 'Sample Library.xel', includes an example of an audit trail form that is used in many of the Metastorm sample procedures.

To add a new form to the library, select **Component | New | Form**. A new form is displayed and added to the **Procedure Explorer**. In a library, forms may be added to map segments or the **Common** item. A form added to the **Common** item does not have access to custom variables. The form may be completed in the usual way.

Refer to section 6 for information on building forms.

Form Segments

Form segments are re-usable collections of form fields that can be used as part of a form.

Many forms created for Metastorm processes contain name and address fields. A form segment containing these fields would make forms requiring this feature quicker and easier to build. The Metastorm sample library, 'Standard Library.xel', contains a form segment that provides a standard banner that can be used with any form.

Form segments cannot be nested i.e. they cannot contain other form segments.

To add a new form segment to the library, select **Component | New | Form Segment**. A new form segment is displayed and added to the **Procedure Explorer**. In a library, form segments may be added to map segments or the **Common** group. A form segment added to the **Common** group does not have access to custom variables. Fields are added to a form segment in the same way as when building a form.

Refer to section 6 for information on building forms.

Map Segments

Map segments can be used for subsections of processes that are repeated. This saves the process designer creating the subsections more than once. They may also be used to break down a large map into several smaller map sections. The main map serves to link all the map segments together.

Map segments:

- Can be nested i.e. they may contain other map segments.

 © Metastorm Inc., 2008

- Can have multiple end points, but only one starting point.
- Can define roles that are accessible by the parent procedure and other map segments.

To add a new map segment to the library, select **Component | New | Map Segment**. A new map segment is displayed and added to the **Procedure Explorer**. You add stages, actions, roles, flags, scripts and custom variables to a map segment in the same way as you would when creating a map.

Refer to section 3 for instructions on creating maps.

Scripts

Client and server side scripts stored in a library may be called from any map or form in the procedure to which the library is associated.

Add a new script to the library in the same way as you would add a script to a procedure.

Refer to section 10 for more details on including scripts in Metastorm BPM.

LDAP Alias

If you want to access directory services data within a procedure, you must create an alias for the connection to the directory within a library.

Refer to section 10.1.7 for further information on direct directory access and adding LDAP aliases.

Integration Wizard Collection

A library may contain a set of additional Integration Wizard functions. This is referred to as an Integration Wizard Collection. Associating a library containing an Integration Wizard Collection with your procedure allows you to use these additional functions in your procedure. The most common use of this feature is to create a number of scripts in a library and provide custom Integration Wizard functions to access these scripts in a user friendly way.

To create custom Integration Wizard functions in the Designer, you will need to add the relevant scripts to a library, and populate the library's Integration Wizard collection. To populate the Integration Wizard collection, you will need to provide an Integration Wizard extensions file. This file is in XML format and contains details of the custom Integration Wizard categories and functions.

For information about the Integration Wizard extension file, refer to the Designer Customization Developer Guide, which is provided with the Metastorm BPM Software Developer's Kit.

The custom Integration Wizard items will be available to any procedure with which the library is associated.

To associate custom Integration Wizard functions with a library:

1. Add any scripts to be called by the custom Integration Wizard items.
2. Create an Integration Wizard extensions file.

3. Add the Integration Wizard extensions file to the library via the Library Properties dialog's **Integration Wizard collection** page.
4. Click on the **Open…** button and select a file to add.
    - If the contents of the selected file are not in the correct format, the file will not load successfully.
    - If the file loads successfully, the custom Integration Wizard categories and items will be displayed in the list.

If an Integration Wizard extensions file has already been added to the library, this will be overwritten when a new one is added. It is possible to associate only one Integration Wizard extensions file with a library. However, the file may contain many Integration Wizard items, and more than one library may be associated with a procedure.

You can export details of existing custom categories and items from a library to a new Integration Wizard extensions file by clicking on the **Save As…** button and choosing a file name.

Flags

Flags are supported in libraries only when they are associated with a map segment. Flags associated with common are not supported.

When a library containing a map segment is associated with a process and that map segment is associated with a map, flags associated with that map segment become available to the map. The flags will not appear in the flag dock window but will appear in the relevant properties of the property editor.

2.1.3 Saving and publishing a new library

When you have added the required parts to the library, save and publish it in the same way as you would save and publish a procedure. A library must be published before it can be used within a procedure. By default, a library is saved with a '.xel' file extension. The number of characters in the filename cannot exceed 31.

2.2 Associating a Library with a Procedure

This section describes how to associate a library with an existing procedure, and how to reference library parts from the procedure.

A library can be updated only when the .xel file is open in the Process Designer. It cannot be updated when a procedure with which it is associated is open in Designer.

2.2.1 Using a library

To associate a library with a new or existing procedure:

1. Open the **Procedure Properties** dialog and select the **Used Libraries** tab.

 © Metastorm Inc., 2008

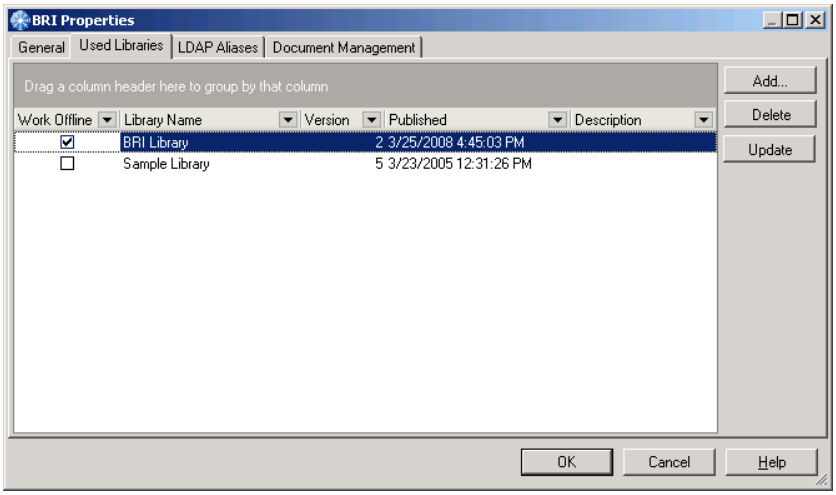


Figure 26: Procedure Properties Used Libraries tab

2. Click on the **Add** button.
    - A database connection dialog appears.
3. Enter the database connection details and click **OK**.
    - The **Add Library** dialog is displayed.

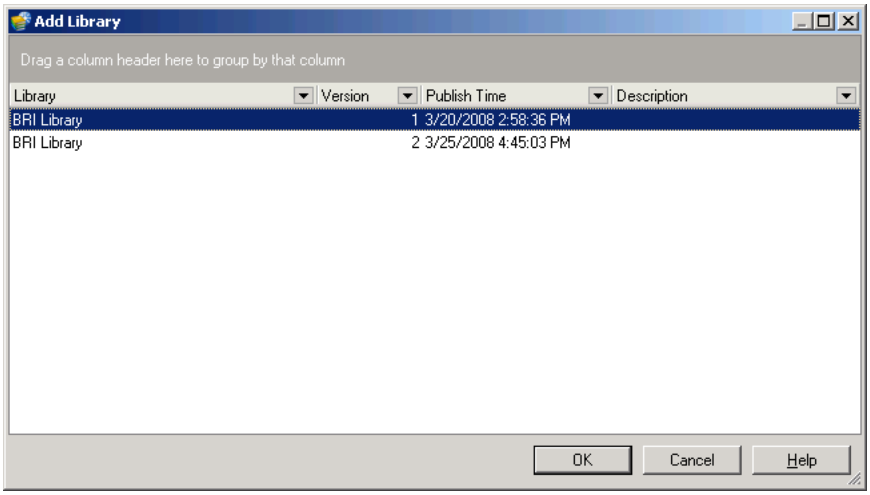


Figure 27: Add Library dialog

4. In the **Add Library** dialog, select the version of the library you wish to associate with the procedure, and click **OK**.
    - The library's details appear in the library list on the **Used Libraries** tab.

The library is bound to a procedure when the procedure is published.

Parts stored in a library become available for use in any procedure with which the library is associated. Set out below are points that should be noted about accessing particular library parts from a procedure.

2.2.2 Accessing a library's forms, form segments, map segments and scripts

Forms, form segments, map segments and scripts from the associated library are available in the same way as those created within the procedure. For example, to associate a form with an action, select the **Form** tab on the **Property Editor** and choose from the list of forms in the **Form** dropdown. This list includes forms from libraries associated with the procedure.

Parts that are held in an associated library use the syntax: <LibraryName>.<PartName>.

When using ***form segments*** from a library, remember that form segments:

- Can be used only once per form
- Cannot be nested i.e. they cannot contain other form segments
- Adopt the same background (color or image) and font as any forms they are placed on.

When using ***map segments*** from a library, remember that:

- Map segments belong to any maps in which they are used.
- Map segments can be used multiple times in a map.
- Custom variables that are defined in a map segment are available to (and are part of) the parent map that the map segment is used in.

  Custom variables that are declared in a map segment will be added to the map table of the parent map.

- Custom variables that are defined in the parent map are not available for use in map segments.
- It is not possible to use custom variables from different libraries that have the same name, but different types.
- Forms, roles, flags and scripts in the map segment are available to any maps in which they are used.
- If you attempt to re-open folder or use chained actions on an action that leads into a map segment stage, ensure that the first stage in the map segment is not a System, Sub-procedure or Archive stage. It is not possible to re-open a folder or chain actions at these stages.

Client and server side ***scripts*** stored in a library may be called from any map or form in the procedure to which the library is associated.

2.2.3 Inheritance of Roles from Libraries

Roles may be defined within a library in either the Common section or in a map segment. When a library is included in a procedure the roles defined within the library become accessible to the other components within the procedure. The accessibility of roles defined in a library by other components within the procedure is as follows:

- Roles defined in the library's common section are added to the procedure's common section and are available to all components within the procedure.

 © Metastorm Inc., 2008

- Roles defined within a library map segment are available only to the components of the procedure maps to which the map segment is applied.

Nested Map Segments

Within a procedure it is possible to have a number of nested map segments, e.g. map segment A applies map segment B which applies map segment C. In such cases there is a hierarchy of inheritance. Each map segment can access the roles of only those map segments applied within it. In the example above, A can access the roles of A, B and C, B can access those of B and C, whilst C can access only those of C.

Recursive Map Segments

It is possible to create a recursive relationship between map segments wherein map segment A is applied in map segment B, and map segment B is applied in map segment A. In this relationship map segment A can access Map section B's roles and map section B can access map section A's roles.

Duplicate Names

When developing a procedure process designers are not restricted in the use of role names. It is possible to duplicate role names within a library with those defined within the parent procedure. If duplicate role names exist, those defined in the parent procedure take precedence. Roles with duplicate names in the library are not accessible.

If duplicate roles exist and the role is removed from the parent procedure the role becomes inaccessible even though it remains in a library. In such a case, the role can be activated in the parent procedure by deleting and reinserting the library.

When a procedure is validated warnings are given whenever a duplicate role name is encountered. Validation occurs and warnings are displayed when the user assigns a library or when a procedure that contains libraries is loaded. *Figure 28* below shows an example of a warning of duplicate role names.

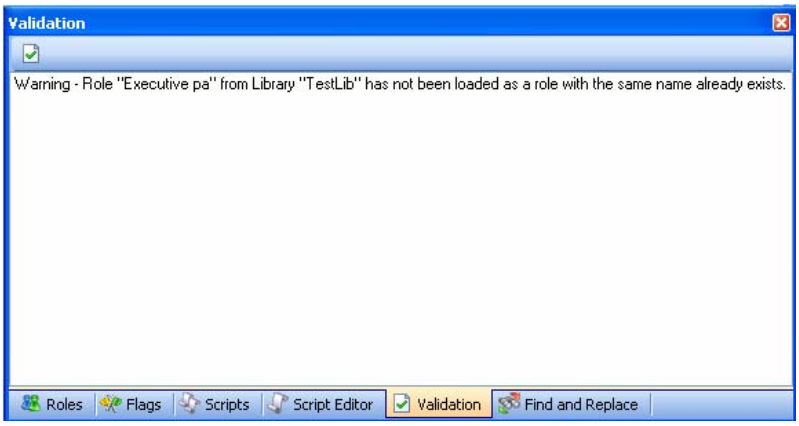


Figure 28 - Warning of Duplicate Roles

In addition, when a procedure is published Designer will warn of any roles with the same name in other procedures in the database.

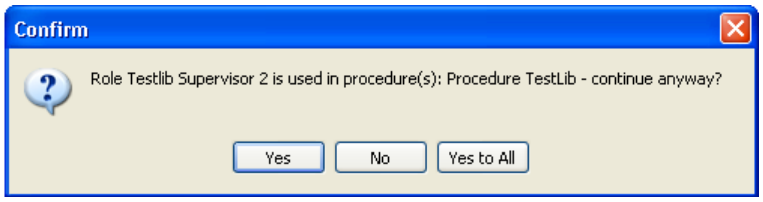


Figure 29: Warning of duplicate roles in another procedure

Variables

Roles defined in libraries can use either system or map variables. When using a library role in a procedure the corresponding variables must be present. Variables defined within a map segment are transferred to the parent map when the map segment is assigned.

Removing Libraries

When a library is removed from a procedure all the Roles defined within the library are removed from the procedure. Similarly, when a map segment is removed from a procedure map, the associated roles are removed from the procedure map.

If changes are made to libraries that remove roles then the roles will not be visible or accessible in the parent procedure.

Role Maintenance

Roles that are made accessible to a procedure by the application of a library cannot be edited from the procedure. Changes to roles must be made by editing the library.

2.2.4 Editing a Library

Libraries can be edited using the **Procedure Properties** dialog to add and remove items. Within the dialog a **Refresh** button is available to implement the changes and reload the procedure.

2.2.5 Accessing a library's LDAP Alias

When a library is associated with a procedure, any LDAP aliases that are set in the library are brought in to the procedure. These are displayed in the **LDAP aliases** tab in the **Procedure Properties**.

2.2.6 Accessing a library's Integration Wizard collection

When a library is associated with a procedure, any Integration Wizard categories and items in the library's Integration Wizard collection are included in the Integration Wizard for that procedure.

 © Metastorm Inc., 2008

2.3 Working with a Procedure that uses a Library

2.3.1 Library Versions

When a library is first associated with a procedure, the required version of that library is retrieved from the Metastorm database. When the procedure is next published, the library components are bound into the process definition in the database.

When you open a procedure that is associated with a library, you are prompted with a **Connection** dialog. You must connect to the database in which the library is stored in order to retrieve the library details for use within the procedure.

- If you cancel the **Connection** dialog, you are given the opportunity to remove the library reference from the procedure. If you choose to remove the reference, you may load the procedure without the library. Without removing the reference, it is not possible to open the procedure without the associated library.

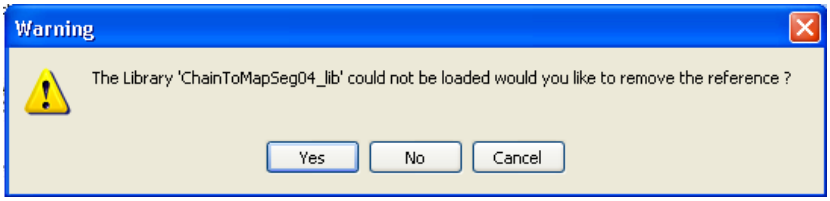


Figure 30: Option to remove library reference

If the procedure is loaded without the library and then saved, references to the library will be lost. Referenced map segments, form segments, folder pages and forms will need to be re associated with stages, folder page lists and actions.

- If you enter database connection details into the **Connection** dialog and press the **OK** button, Designer loads the correct version of the library from the database.

2.3.2 Loading the required version of the library

It is not possible to:

- Store more than one library with the same name on the same database.

It is possible to:

- Store more than one version of a library with the same name on the same database.
- Store more than one library with the same name on different databases.

Therefore, Designer distinguishes between different versions of different libraries by checking the library name, the version number and a library identifier.

When you open a procedure, Metastorm Designer searches the selected database for the libraries that have previously been associated with the procedure via the **Procedure Properties**.

For each associated library:

- **If a version of the library with a matching version number** is found, the Designer checks for additional versions of the same library:
  - If no other version is found, the library is loaded, and you are notified by the following **Library Loading Status** dialog.

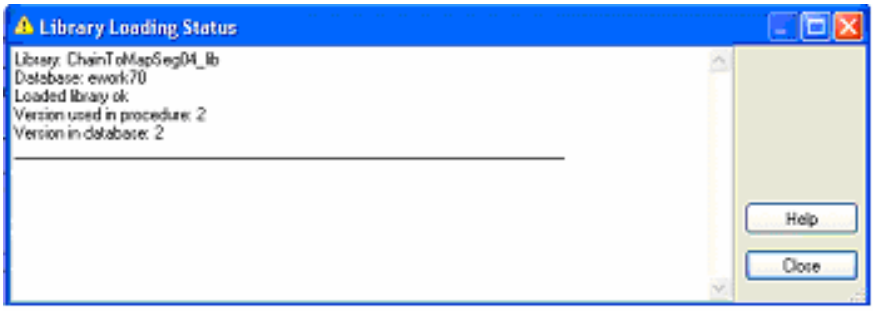


Figure 31: Referenced library loaded

  - If another version of the same library is also found, you are prompted to choose which version to use.

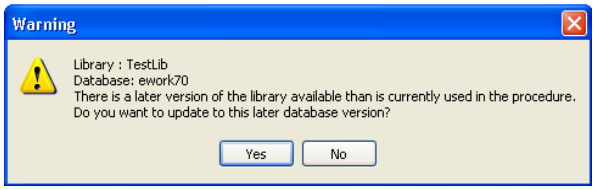

Warning
Library : TestLib
Database: ework70
There is a later version of the library available than is currently used in the procedure.
Do you want to update to this later database version?
Yes
No

Figure 32: Later version available

  - If you click on the **No** button, the version that is referenced in the procedure is loaded and used.

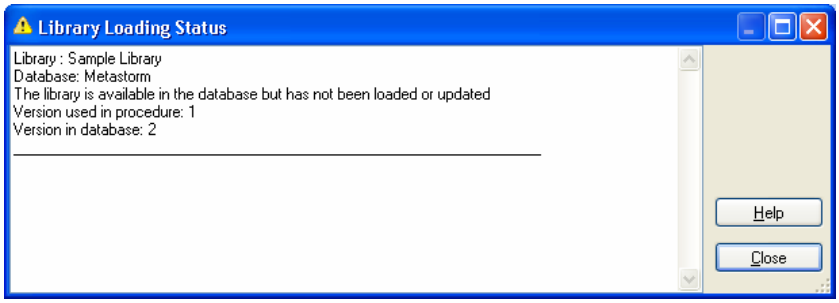

Library Loading Status
Library : Sample Library
Database: Metastorm
The library is available in the database but has not been loaded or updated
Version used in procedure: 1
Version in database: 2
Help
Close

Figure 33: Referenced version loaded

  - If you click on the **Yes** button, the latest version is loaded and assigned to the procedure. Next time the procedure is opened the Designer will attempt to open this version of the library.

 © Metastorm Inc., 2008

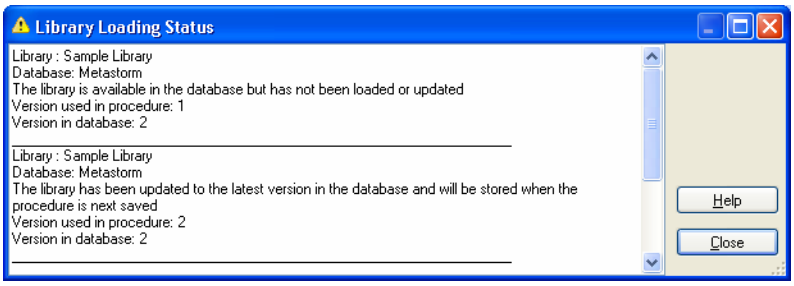

Library Loading Status
Library : Sample Library
Database: Metastorm
The library is available in the database but has not been loaded or updated
Version used in procedure: 1
Version in database: 2
Library : Sample Library
Database: Metastorm
The library has been updated to the latest version in the database and will be stored when the procedure is next saved
Version used in procedure: 2
Version in database: 2
Help
Close

Figure 34: Latest library loaded

- If another version of the same library, with identical contents, is found the latest version is automatically loaded.

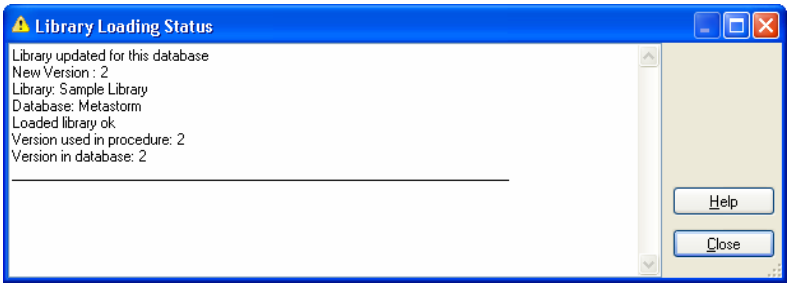


Figure 35: Latest library automatically loaded

- **If a different version of the library** is found, you can:
  - Use the latest version that exists in the database
  - Delete the reference to the library from the procedure
  - Search another database for the correct version.

You are presented with the following **Warning** dialog:

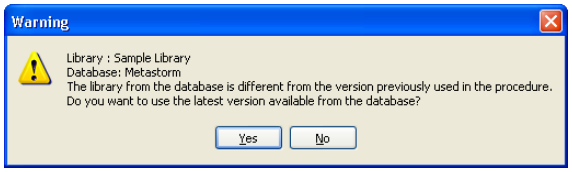

Warning
Library : Sample Library
Database: Metastorm
The library from the database is different from the version previously used in the procedure.
Do you want to use the latest version available from the database?
Yes
No

Figure 36: Option to use latest existing version

If you click on the **No** button, the library is not loaded and the procedure is not opened.

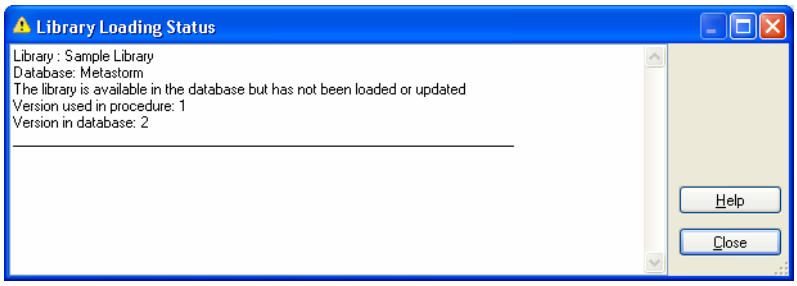

Library Loading Status
Library : Sample Library
Database: Metastorm
The library is available in the database but has not been loaded or updated
Version used in procedure: 1
Version in database: 2
Help
Close

Figure 37: Latest existing version not loaded

You may search another database, or remove the reference to this library from the procedure. If the reference to the library is removed, the procedure can be opened, but may not function as intended.

If you click on the **Yes** button, the latest existing version of the library is loaded and the procedure is opened.

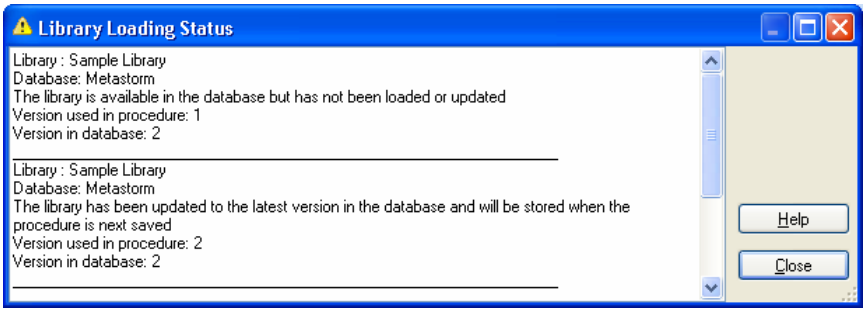

Library Loading Status
Library : Sample Library
Database: Metastorm
The library is available in the database but has not been loaded or updated
Version used in procedure: 1
Version in database: 2
Library : Sample Library
Database: Metastorm
The library has been updated to the latest version in the database and will be stored when the procedure is next saved
Version used in procedure: 2
Version in database: 2
Help
Close

Figure 38: Latest existing version loaded

- **If no version of the library is found**, you may choose another database to search, or choose to remove the reference to the library from the procedure.

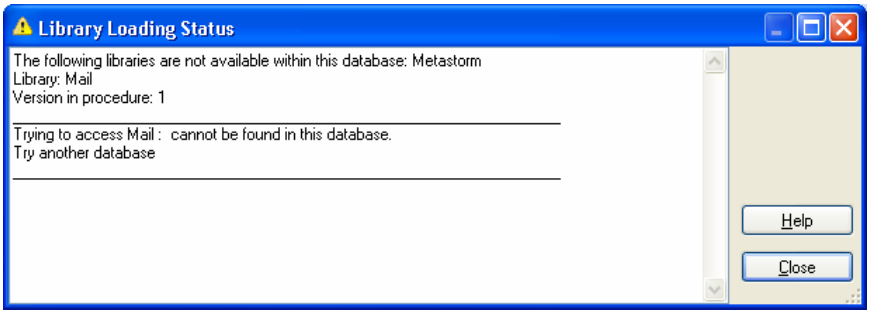

Library Loading Status
The following libraries are not available within this database: Metastorm
Library: Mail
Version in procedure: 1
Trying to access Mail : cannot be found in this database.
Try another database
Help
Close

Figure 39: Library not found

Summary

The following table summarizes the scenarios described above:

Scenario	*Outcome*
A library with a matching version number is found.	The library is loaded and the process designer is notified.
A library with a matching version number is found. Another version of the library also exists in the database.	The process designer is prompted to choose whether to use the current version or the latest one.
A library with a matching version number is found. A version of the library with a different version number but identical contents also exists in the database.	The latest version of the library is automatically loaded.

 © Metastorm Inc., 2008

Scenario	Outcome
A version of the library is found, but the version number does not match that which is stored in the Procedure Properties.	The process designer may choose to use the latest version that exists in the database. If he chooses not to use this library, he may choose to delete the reference to the library from the procedure. Alternatively he may choose another database in which to search for the correct version.
No version of the library is found in the database.	The process designer may choose another database to search, or choose to remove the reference to the library from the procedure.

Table 11: Loading a library version

If versioning is switched off in the Library, the version number will not be updated when the library is saved. However, an additional internal library identifier is updated. This identifier is taken into account when loading the library with a procedure.

2.3.3 Reverting to an Earlier Library Version

It is possible to update a procedure so that it uses an earlier version of a library. To do this, first delete the reference to the library in the **Procedure Properties** dialog. Next, re-associate the library by adding the required version.

2.3.4 Working Offline

It is possible to work on a procedure when the associated library is offline. An offline library is one that is accessed from a file rather than from the Metastorm database. This feature allows process designers to work on procedures without access to the Metastorm database.

To take a library offline click on its corresponding **Work Offline** checkbox in the procedure's **Used Libraries** tab.

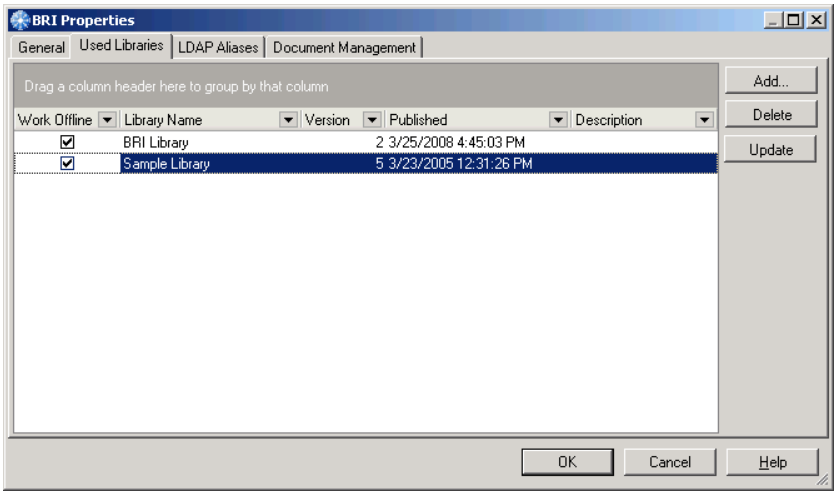


Figure 40: Work offline checkbox

To take a library offline

1. Open a procedure file that has a library associated with it. **Select File | Procedure Properties** and select the **Used Libraries** tab
    - The **Work Offline** checkbox should be unchecked at this stage.
2. Select the library file in the libraries list.
3. Check the **Work Offline** checkbox.
    - Designer takes a copy of the library file that is loaded and writes it to the designated file path.

*To set the file path, choose **View | Options** and enter the path into the **Library Path Location** on the **File Locations** tab.*

4. Click **OK** to close the Procedure Properties dialog and close the procedure.

When the procedure file is reopened with the **Work Offline** checkbox checked, the library file is read from the file in the designated file path. If the library file cannot be found, either delete the library from the procedure or read the library from the Metastorm database.

If the procedure is loaded without the library and then saved, references to the library will be lost. Referenced map segments, form segments, folder pages and forms will need to be re associated with stages, folder page lists and actions.

To take a library online

1. Open the procedure.
    - The procedure Loading Status dialog will indicate that the library is from the offline source.

Figure 41: Library Loaded from offline location

2. Select **File | Procedure Properties** and uncheck the **Work Offline** checkbox.
    - The following dialog is displayed:

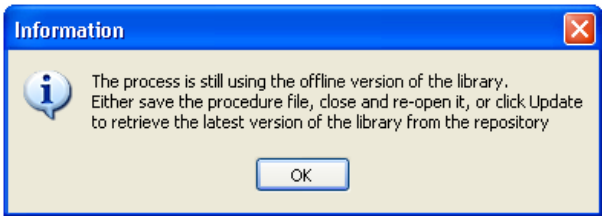

Information
The process is still using the offline version of the library.
Either save the procedure file, close and re-open it, or click Update
to retrieve the latest version of the library from the repository
OK

Figure 42: Taking a library online

 © Metastorm Inc., 2008

- Designer will continue to read the library from the designated source.

3. Click on the **Update** button on the **Procedure Properties Used Library** tab to read the library from the Metastorm database.
4. Save and close the procedure file with the **Work Offline** checkbox unchecked.

When the procedure file is opened again the library is read from the Metastorm database.

Updating the library offline

While the library is offline, you may open the file in the Designer, make updates and save the file in the offline location.

To make these changes available online, you must publish the library to the database before going back online. If the library is taken online without publishing the updates, the procedure will revert to the original version of the library in the database that was taken offline.

3 MAPS

The Designer uses one or more maps to represent the steps required to complete a business process. A map describes graphically how to process a folder of a particular type. A map consists of a number of stages where the folders await some action. In addition a procedure may include system stages where folders are acted upon automatically by the system. Stages are linked by actions.

A map contains the following elements:

- A single Start Box, from which new folders are created.
- One or more stages, where folders are held pending one of a number of possible actions; typically, a stage represents either the system or one or more users, who must perform a particular task.
- One or more actions which update the contents of a folder and move the folder to the next stage.
- Any number of comments.

Maps are built by creating stages and linking them with the actions necessary to move the folder to the next stage.

The Properties editor is used to:

- Name the stages and actions; and
- Assign folders to To Do and Watch lists;

Map segments are sections of maps that can be re-used in other maps. Map segments are designed in the same way as maps but with the following restrictions:

- There is one entry point and there may be multiple exit points; and
- The first action in a map segment has no properties and is not removable.

 © Metastorm Inc., 2008

Apart from these issues, designers can create map segments in the same manner that they create maps. Map segments can be saved independently from the procedure file and added to other procedures.

3.1 Creating a New Map

Before using the Designer for the first time, you should be familiar with the terminology used and have a basic understanding of automated business process design.

Whenever you start Designer or create a new map, a default map is displayed in the Main Pane. This map consists of a Start Box and a User stage connected by a User action.

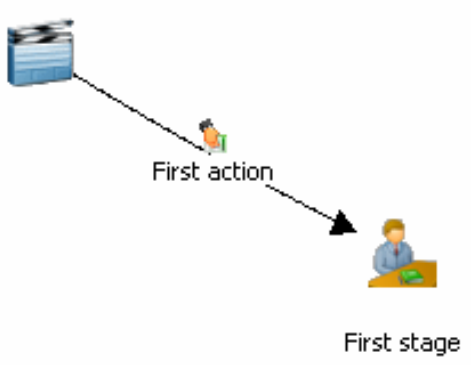


Figure 43: Default Map

As you add elements to a map, you may find it necessary reorganize the elements to optimize the space. Designer provides a set of tools to assist you in map layout and design.

3.1.1 Working with Map Elements

Adding Stages to a Map

To add a stage, click on the desired stage button on the Map toolbar and then click on the map to place the stage. Stages can be added and then changed using the Properties editor. Stages may be moved around on the map by selecting them and dragging them to the desired location.

Stage names can use characters which are configured with a default locale of North American, Western European or Central European character sets. However, stage names cannot include any of the following characters: Pipe (|), Period (.) Dollar Sign ($), Equal Sign (=), Back Slash (\), Quotation Mark (“), Forward Slash (/), Percent (%), Comma (,), Plus (+), Colon (:), Semi-colon (;) and Angle brackets (<>).

When designing a stage to expose as a web service, the Stage name should contain only alphanumeric characters and spaces.

Adding Actions to a Map

To add an action, click on the desired button on the Map toolbar and then click on the stage that represents the source of the action. Hold down the left mouse button and drag the action from the

'source' stage to the 'destination' stage. If the action loops back to the same stage, drag the action out from the stage and release the mouse button.

To add several copies of a particular map element, hold down the Shift key while clicking on the desired element button, then click at each location on the map where you want the element to be placed. Click on the Pointer button or select another map element to disable the multiple placement function.

Adding Comments to a Map

Any number of comments can be placed on the map to provide additional information for designers. To add comments

1. Click on the **Comments** button on the toolbar, then click on the Map where the comment is to be placed.
    - A shaded box with a shadow outline is placed on the Map.
    - The Properties editor is displayed , and
    - A comments box is added to the Procedure Explorer.
2. In the Properties box scroll through the available drop-down list and select the font in which the comments are to be displayed.
3. Click the font format button ;
    - The standard Font dialog is displayed.
4. Select the font, text color and size required.
5. Type the comments in the Comments field.
    - The comments are displayed in the Comments box on the map.

The Comments box can be repositioned by selecting it and dragging it to the desired location. The Comments box may be resized as necessary.

Moving Map Elements

To move a map element:

- Select the element by clicking on the **Pointer** button on the Map toolbar and then clicking on the map element. Click anywhere on the element and drag it to its new position.

To move multiple elements select the elements and drag and drop the group selected. To select the group of elements to be moved:

- Select the **Pointer** button. Place your cursor over a point on the map, click on the left mouse button to change the cursor to a crosshair, and, still holding the left mouse button down, drag your cursor to encompass all of the desired map elements. Designer will display a dashed line indicating the outer edges of your selection. All items within the dashed line will be selected when you release the mouse button.
- Select the **Pointer**, button, and click on a map element. Hold either the Shift or the Ctrl keys, select any additional elements desired.
- From the Edit menu choose **Select All**. Designer will select all elements currently on the map.

 © Metastorm Inc., 2008

If you have selected multiple map elements, the handles surrounding each selected element will be disabled. You may also cut, copy or delete the elements as a group.

Moving Map Elements

Map elements may be moved around on the map by selecting them and dragging them to a new location.

When map elements are repositioned the connecting actions lines will be re-routed.

Copying and Pasting Map Elements

Map elements may be copied by:

- Selecting the desired element and then selecting the **Copy** button from the General toolbar, or
- Using the **Edit | Copy** option.

Map elements may be pasted onto a map by

- Selecting the **Paste** button from the General toolbar, or
- Using the **Edit | Paste** option.

Determining Map Element Alignment

Individual and groups of map elements may be centered horizontally or vertically on the map, and multiple map elements may be aligned vertically or horizontally relative to each other through the **Edit | Align** option. The following table shows the selections available through this option:

HORIZONTAL	*VERTICAL*
No change	No change
Left sides	Tops
Centers	Centers
Right sides	Bottoms
Space equally	Space equally
Center in window	Center in window

Table 12: Map Element Alignment

Determining Map Element Layering

Overlapping map elements should be avoided as it makes the map difficult to read and may make individual elements difficult work with. However, as you create a map you may find that you need to place one map element in front of or behind another.

To place one map element *in front* of another, select the map element you want to bring to the front, and select **Edit | Bring To Front**. Alternatively, select the map element, right-click, and a pop-up version of the **Edit** menu will open.

To place one map element *behind* another map element, select the map element you want to send to the back, and select **Edit | Send To Back**. Alternatively, select the map element, right-click, and a pop-up version of the **Edit** menu will open.

The icons used to create a map consist of an image and a caption on a transparent background. Layering map elements too closely together may result in confusion if the captions are not aligned properly. You should inspect your map layout to verify that the captions for each map element appear as desired.

Map element layering affects windowed controls (such as frames) and non-windowed controls (such as labels) separately. So, you cannot change the layering of labels (the only non-windowed controls used in the Designer) relative to other (windowed) Designer controls.

Removing Elements from a Map

Map elements may be deleted from a map using one of the following methods:

- Select the desired element and press the **Delete** key on your keyboard.
- Select the desired element, and use the **Edit | Delete** menu option.
- Select the desired element, and use the **Edit | Cut** option.
- Select the desired element, and click on the **Cut** button.

Deletion of a stage will also delete all of the actions connected to that stage.

*Map elements removed using either of the **Cut** methods (**Edit | Cut**, or the **Cut** button) are saved in the paste buffer, and may be pasted back onto another map in the procedure. Map elements deleted using the **Delete** options (**Edit | Delete**, or the* Delete *key) are not stored. However, you may use the **Edit | Undo** option to restore deleted items immediately.*

3.2 Properties editor

Each of the elements of a map or map segment has certain default properties; usually it will be necessary to modify some or all of these properties. This is done through the Properties editor.

 © Metastorm Inc., 2008

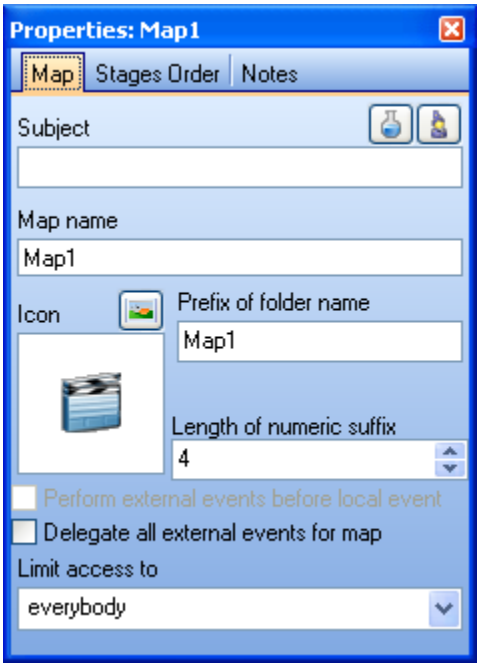


Figure 44: The Properties editor

The Properties editor may be floating or docked to the top or sides of the main pane. To dock the Properties editor, position your cursor anywhere in the title bar and drag it to the far right, far left or top of the main pane. Once the Properties editor has been docked it can be set to appear and collapse by selecting the auto-hide feature using *Push-Pin* on the Properties title bar.

The Properties editor is initially a floating dialog which you may move or toggle on or off. To:

- Dock the Properties editor, click on the Properties editor title bar and drag it to the far right side of the Main Pane.
- Open/Close the Properties editor, select the Properties button from the General toolbar. Alternatively, you may toggle it on/off by using the menu option **View | Properties** or by selecting Properties option from the popup menu that appears when you place your cursor in the Main Pane and right-click.

When you start the Designer, the Properties editor will automatically be displayed.

Each map element has a Properties editor tab specific to that type of element. Other tabs of the Properties editor are common to several different map elements, but may display different property fields, or require instructions based on the type of map element you are working with. This section is organized by map elements, and includes instructions for working with all Properties editor tabs for each type of map element individually.

Many of the properties you will modify through the Properties editor require you to enter a formula. In most cases these fields will include options for calling on the Integration wizard and the Formula editor.

Instructions for using these tools may be found in Section 10:Creating Formulas.

3.2.1 The Start Box

The Start Box is represented by the clapper board icon on a map. It is a default element placed on the map representing the starting point of the procedure.

There are several restrictions applied to Start Boxes:

- You can move the Start Box within the map but you cannot remove it from the map.
- You cannot draw conditional, timed, or rendezvous actions from the Start Box.
- You cannot draw an action to the Start Box.

Creation Actions

Actions that leave the Start Box are called creation actions.

You can draw any number of user creation actions and any number of flagged creation actions. Each creation action may be associated with a different form and may or may not go to the same stage.

If you want to use different forms to initiate the same map (for example, for different levels of staff), use a separate creation action for each form.

3.2.2 Defining Map Properties

The Properties editor displays three tabs for defining Map properties:

- Map
- Stages Order
- Notes

Defining a Map's Map *Properties*

 © Metastorm Inc., 2008

Figure 45: Map Tab

1. Click on the icon representing the Start Box on the map, or any place in the Main Pane when the map is displayed.
2. Enter the name for the map in the **Map name** field. This name must be different from the name of any other map to be published to the same database and must begin with a letter. A map name may not exceed 20 characters nor contain extended characters or any characters other than letters, digits, underscores or spaces.

A number of words have special meanings in the databases that Metastorm BPM supports. You should not use any of these reserved words when naming a map. See the Reserved Words.txt file for a list of reserved words. This file is located in the Metastorm BPM \Designer folder.

You should not use system variable names when naming a map. A list of system variables is contained within the oemdesigner.xml file under the <Variables></Variables> node. This file is located in the Metastorm BPM \Designer folder.

Oracle object names should not be used for map names.

*When designing a map to expose as a web service, the **Map name** should only contain alphanumeric characters and spaces.*

3. Metastorm BPM automatically generates a name and number for each folder created by a procedure, e.g. "Training Request 0057."

   To specify the prefix the system should use for this procedure's folders, enter the value in the **Prefix of folder name** field. There is a 20-character maximum for the Map Prefix. You may define several types of prefixes. For example:

   - A short phrase – such as "Training Request" for a training request procedure.
   - A keyword – such as "Travel" for a travel request procedure.
   - An alphanumeric code – such as "MP" for a motor pool resource allocation procedure.

   Whatever type of prefix you select, it should easily identify the procedure that generated the folder when the folder appears in a user's *Watch* or *To Do* list. If you do not specify a value, Designer uses the map name.

   If you change the Prefix of a folder name *property after the procedure has been published and used, the numbering of subsequent folders created using this procedure will be restarted at 1.*

4. You can specify the size of the auto-incrementing numerical suffix used when generating a name for new folders created through this procedure. There is a maximum of 9 characters for the map suffix.

   Enter the number in the **Length of numeric suffix** field. Designer will use pad-out the field with leading zeroes if fewer digits are entered, e.g. MP0032.

   To suppress the numeric suffix, enter 0. To set the numeric suffix to contain no leading zeros, enter 1. If you do not supply a value, Designer uses the default of 4.

5. If you want to change the icon used to represent the Start Box on the map, click on the **Image** button. Or double-click on the displayed icon. Designer displays the standard File Open dialog, and you can select a file containing an alternate icon.

6. Type a **Subject** line for the map. Whatever you type here becomes the default subject for each folder created within this map.

   Since no map variables exist for a folder until the first form is committed, if you wish to include a Metastorm variable in the Subject you must assign the subject value in the When action completed *property of the first action in the procedure.*

7. To restrict the users who may locate and access a folder to those with a particular role, select the role from the **Limit access to** drop-down list. In order to prevent unauthorized access, identify the roles that may have access to the forms and folders belonging to this procedure.
8. To be able to create global custom extensions you can use **Delegate all external events for map** which defines whether all the events of a process will be delegated to an external .NET assembly. **Perform external events before local events** defines when the external events are run.

*To use the **Delegate all external events for map** check box the **Process Events Library** needs to be associated with the current procedure.*

For further information on creating local custom extensions refer to Appendix B, page 392 or Process Orchestrator for .NET Designer's Guide.pdf

Defining a Map's Stages Order *Properties*

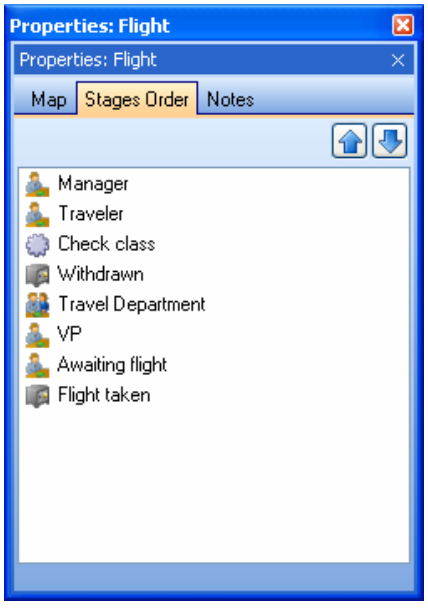


Figure 46: Stages Order Tab

The **Stages Order** property enables you to establish the stage priority for each stage on the map. The stages pass their priority to the actions originating from them. The **Stages Order** property is especially important when a common stage with user actions or conditional actions originating from it is applied to another stage on the map with user or conditional actions originating from it.

For a set of user actions originating from a single stage, the **Priority** property for each action determines the left-to-right display order of the actions in the user interface. The action with the

lowest priority (1) is displayed in the left-most position. Applying this to the **Stages Order** property, if a user stage has a higher place in the **Stages Order** property than a common stage, its actions will take precedence over those of the common stage and will be displayed farthest to the left in the Metastorm client.

For a set of conditional actions originating from a single stage, the sequence in which the actions are assessed is determined by the action's **Priority** property. On more advanced maps you may encounter conditional actions originating from the stage itself and conditional actions applied from a common stage. On these more advanced maps, the **Stages Order** property contributes to determining the order of assessment for the conditional actions. If the common stage is higher on the **Stages Order** list, the conditional actions from the common stage will be assessed before the conditional actions on the other stage.

The default order of the stages on the Stages Order tab is defined by the order in which the stages are originally added.

To change the Stage Order for stages on your map, select a stage from the list, and use the up and down arrow buttons to move the stage to its new position, or drag and drop the items.

Defining a Map's Notes *Properties*

This tab of the Properties editor displays a memo field that the designer may use to make notes. These notes are not available to the end user.

3.3 Stages

When a folder reaches a user's desktop, Metastorm BPM views the folder as having reached a *stage* in the procedure. A map may also contain various *system stages* that do not require human interaction.

You can include any number of stages on the map. Stages are connected by actions, which will be discussed later in this section. To add a stage to a map, select the desired stage button from the Map toolbar and click on the map to place the stage. Designer places the appropriate stage icon on the map and displays the corresponding Properties editor. Stage icons may be moved around on the map dragging and dropping them to the desired location.

To add several copies of stage, hold down the Shift key before clicking on the desired Stage button.

There are six types of stages available within the Designer. The following table summarizes the stages, their purpose, rules of use, and the Properties editor tabs associated with each:

Stage	*Purpose*	*Rules*	*Properties editor Tabs*
User	Folders at a user stage will generally be placed on a single user's To Do list.	You can draw any type of action to a user stage. You can draw any type of action except a rendezvous action from a user stage.	Stage Roles Do this Notes

Stage	Purpose	Rules	Properties editor Tabs
Group	Folders in a group stage will generally be placed on several users' To Do lists.	You can draw any type of action to a group stage. You can draw any type of action except a rendezvous action from a group stage.	Stage Roles Do this Notes
System .	Folders at a system stage will not be placed on any users' To Do list. Folders in this type of stage are handled automatically	You can draw any type of action to a system stage. You can draw any type of action except a rendezvous action from a system stage.	Stage Roles Do this Notes
Sub-Procedure	Allows you to divide a large map into a number of sub-maps that can be processed in parallel. The folder that reaches the sub-procedure stage will give rise to a number of sub-folders (child folders), each of which will follow the flow of its own map.	You can draw any type of action to and from a sub-procedure stage.	Stage Roles Maps Do this Notes
Rules	Allows you to execute a rule using a Business Rules Engine. Although you can also use the %ExecuteRule function in a standard 'When…' event, the advantage of executing a rule from the **Rules** property is that the rule is executed after the form is submitted, rather than as part of Submit processing, so that control is returned to the client sooner.	You can draw any type of action to a rules stage. You can draw any type of action except a rendezvous action from a rules stage.	Stage Roles Do this Notes
Common	A common stage represents a list of other stages. You use this stage type to define common actions that should be available from each listed stage.	You cannot draw an action to a common stage from another stage, although loopback actions are allowed. You can draw any type of action except a rendezvous action from a common stage.	Stage Do This Applied to Notes
Archive	No further processing is expected. Folders at an archive stage will not appear on any users' To Do or Watch lists.	You can draw any type of action to an archive stage. You cannot draw any type of action from an archive stage.	Stage Do this Notes
Map Segment .	Allows you to reuse common sections of a map. When a folder reaches a Map segment stage, it enters the specified map segment. When it comes to the end of the map segment, it re-joins the main map. *If a formula added to the rendezvous action coming from a map segment stage fails, the folder will remain at the map segment stage*	You can draw any type of action to a Map Segment stage. You can only draw a rendezvous stage from a Map Segment.	Stage Do this Notes

 © Metastorm Inc., 2008

Stage	Purpose	Rules	Properties editor Tabs
Comments	Enters text boxes onto procedure maps for the addition of comments.		

Table 13: Stages

There are two types of property tabs you can define for stages:

- stage-independent tabs; and
- stage-dependent tabs.

3.3.1 Stage-independent Tabs

The following stage-related properties tabs are common to all stages:

- Stage.
- Roles.
- Do this.
- Notes.

Defining Stage *Properties*

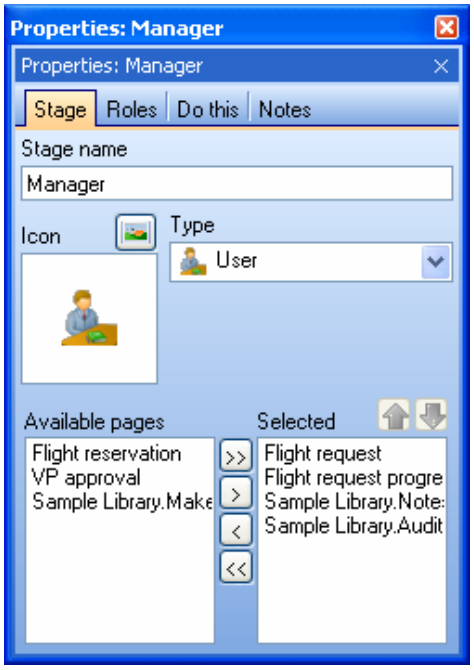


Figure 47: Stage Tab

Enter the name of the stage in the **Stage name** field. This will be the name displayed on an end user's *To Do* or *Watch* list. This field must not be left blank, and each name in the map must be

unique. The field may be no longer than 31 characters in length. The following characters are not supported in the **Stage Name** property:

- Pipe (|).
- Period (.).
- Dollar Sign ($).
- Equal Sign (=).
- Back Slash (\).
- Quotation Mark (“).
- Forward Slash (/).
- Percent (%).
- Comma (,).
- Plus (+).
- Colon (:).
- Semi-colon (;)
- Open bracket (<).
- Close bracket (>).

*When designing a stage to expose as a web service, the **Stage name** should contain only alphanumeric characters and spaces.*

The stage type is displayed in the **Type** drop-down list. To change the type of stage you are working with, select the new type from the list.

If you change the stage type after you have defined stage properties, be sure to review the properties for the new stage type.

A stage on the map is represented by the **Icon** shown in the Properties editor. You can change this icon to one of your own choice. To change the stage icon:

1. Double-click on the displayed icon, or by clicking on the **Image** button:
    - The **File Open** dialog is displayed.
2. Browse to select a graphic file containing an alternate icon.

Select the forms that will be displayed to the user while the folder is at this stage from those listed in the **Available pages** field. Use the arrow buttons to move forms into the **Selected** field. Only common forms and forms associated with the current map are listed. The system updates this list as new forms are created within the procedure. Forms will be visible to a user only if they hold the appropriate roles. To change the order in which the folder pages are displayed, change their order in the **Selected** list using the arrow buttons.

*A common stage has no **Folder pages** property.*

Because folders at the Archive stage do not appear on any user’s To Do or Watch list, the folder pages you select here, for an Archive stage, will be visible only through the folder-on-folder feature. This feature allows you to open a folder at its current stage from another procedure, even if that folder has already been archived. The New Hire sample procedure (Designer\Sample Procedures\New Hire.xep) illustrates this feature.

Defining Roles Properties

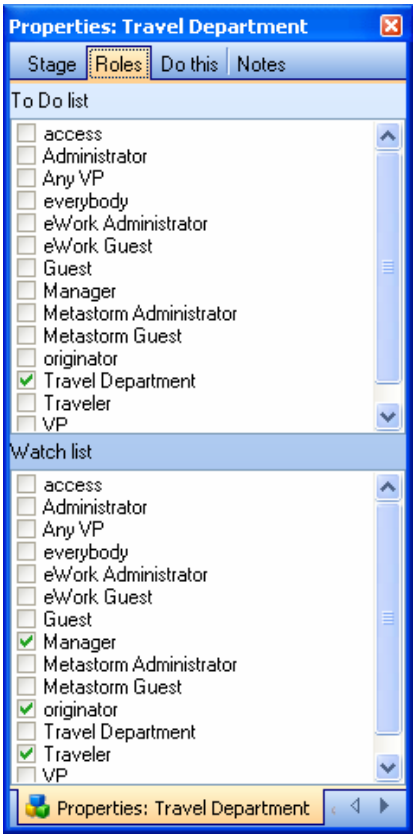


Figure 48: Roles Tab

Common and archive stages have no Roles tabs.

The role options displayed on the **Roles** tab of the Properties editor are determined by the roles listed on the Roles dialog of the current procedure.

For more information on defining roles, refer to Section 7, Defining Roles or the Users and Roles chapter in the Metastorm Administration Guide.

The owners of these roles are determined by the Metastorm Engine when the folder is processed. They are based on the role assignments made in Users and Roles or by the formula defined for the role in the Roles dialog.

The roles selected in the To Do list property and Watch list property on the Roles tab determine on whose To Do and Watch lists the folders will appear. For the To Do list, the default role for a stage is originator. For a Watch list, by default no roles are selected.

Folders at a System stage will not be placed on any users' To Do list; however, they may appear on users' Watch lists. To set up folders to be available on the To Do and Watch lists:

1. In the **To Do list** property, select those roles that should have folders appear on their *To Do* list at this stage in this procedure.

2. Select those roles that should have folders at this stage appear on their *Watch* list from the list in the **Watch list** property.

The same roles may not have a folder appear simultaneously on both its Watch *list and* To Do *list. If you select a role in one field, it will be automatically deselected from the other field.*

If a user holds a role on both the To Do and Watch lists, he will only see the folder on the To Do list as its setting will take precedence.

When a folder arrives at a stage, the Metastorm Engine generates 'alerts' for each user that should see the folder on his To Do or Watch list. Since this can be a relatively time consuming process, the Engine prepares these alerts on a separate thread—thus allowing it to process other tasks simultaneously. This approach avoids performance degradation; however, there are drawbacks. For example, if immediately after arriving at a stage the %ToDoList or %WatchList functions are used, the functions may reflect out of date information (i.e., the users on the To Do and Watch lists from the previous stage.) The manual workaround to this is to apply the same roles as are specified in the originating stage. However, this won't be available from a common stage. In the case of a loop-back action with the 'Remove from ToDo list' option, this could be handled via an additional %ExecSQL to delete the entry from the eAlert table.

Defining Do This *Properties*

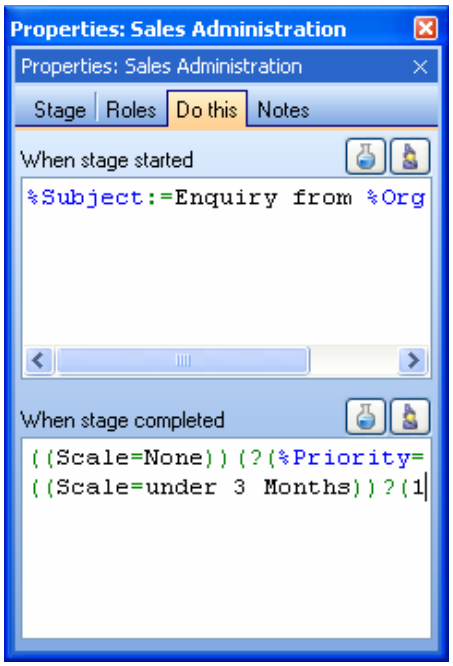


Figure 49: Do This Tab

This tab is used to specify any processing to be carried out at this stage. To specify processing to be carried out when a folder reaches this stage from another:

1. Enter one or more commands into the **When stage started** field. Commands are evaluated when the folder arrives at this stage.

   When stage started *and* When stage completed *commands added to a Common stage are applied to all stages to which the common stage is applied.*

 © Metastorm Inc., 2008

*If a common stage with a **When stage started** formula is applied to another stage containing a **When stage started** formula, the order that the formulas will be evaluated in is determined by the **Stages Order** map property.*

2. Click on the **Formula editor** button to open the Formula editor, or click on the **Integration wizard** button to use the Integration wizard to create the command.

For more details on using the Integration wizard, refer to Section 10: Creating Formulas.

If commands fail to complete successfully, whether on entering or exiting the stage, the action is aborted and subsequent commands will be skipped. Metastorm BPM will log an exception in the Designer Log (which may be viewed through the Services Manager), and the action moving the folder will be aborted, leaving the folder at its original stage in its original state. Any Metastorm database commands that have already been executed will be rolled back. However, any other commands that have already been executed (including commands which update external databases, send emails, print files, write files, or raise Metastorm flags) will not be undone.

3. If you are using a Rules stage, use the %ExecuteRule function to specify, in the **Rules** field, any rules to be executed.

*Although you can also use the %ExecuteRule function in a standard '**When…**' event, the advantage of executing a rule from the Rules property of a Rules stage is that the rule is executed after the form is submitted, rather than as part of Submit processing, so that control is returned to the client sooner.*

4. To specify processing to be carried out when the folder leaves the stage for another, enter one or more commands in the **When stage completed** field.

*An archive stage has no **When stage completed** property.*

5. For a system stage, the **Auto-forward folders to** field allows you to specify a new stage to which a folder should proceed immediately after entry to this stage, regardless of any actions leaving the stage. You can use this property to allow end users to specify the next stage for a folder. The default is for this property to be empty which causes the folder to remain at the System stage.

Do not use Auto-forward to move a folder to another map segment. This interferes with the Engine's ability to determine where the folder should re-enter the current map segment when it is archived in the target map segment.

The value in this property can be a variable evaluated for each folder at this stage. For example, you may wish to put a drop-down list of possible stages on a form, and use that form in the action leading up to the system stage. You would then have the option to use the variable mapped by the drop-down as the target stage.

Auto-forward is implemented by a hidden conditional action named "=>" targeted at the relevant stage. If this hidden action resolves to a loopback action (that is, the target is the same system stage), the Engine will detect this and prevent the auto-forward taking place. A message to this effect will be written to the Application event log.

Custom extensions can be used if the Process Events Library is associated with the procedure. For further information refer to page 392.

*If custom extensions are used, the **Delegate all external events for map** check box in the **Map Properties** tab will determine the order of the **Do This** processes.*

For further information on creating custom extensions refer to Process Orchestrator for .NET Designer's Guide.pdf

Defining Notes *Properties*

This tab of the Properties editor displays a memo field which the process designer may use to make design notes. These notes will not be seen by end users.

3.3.2 Stage-specific Tabs

Defining a Sub-procedure Stage's Maps *Properties*

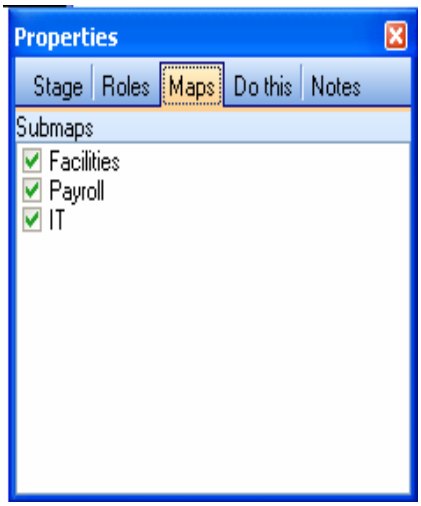


Figure 50: Maps Tab

Sub-procedures are used to open a number of sub-maps, each of which will create a "child folder" that can be processed simultaneously or in parallel with the main procedure.

Sub-procedures should be used only when there is a requirement for the parent folder to wait until the child folders have finished processing. If this requirement does not exist, the procedure should raise a flag to start a new folder in another map or procedure.

In addition to opening parallel procedures that have no direct impact on the parent procedure, Sub-procedure stages may be used in conjunction with Rendezvous actions to place the parent procedure "on hold" status until one or more sub-procedures are completed.

Each time a folder arrives at a sub-procedure stage, a child folder is initiated in each sub-procedure map. Because of this inherent functionality, loop-back actions may not be desirable for sub-procedure stages. Each time a loop-back action is triggered a new set of child folders will be created.

The *Maps* tab contains a checklist of all maps defined for this procedure. The system updates this list as new maps are created within the procedure.

 © Metastorm Inc., 2008

Select the sub-maps you wish to open at this point from the list displayed in the Submaps field. You may check all the maps that apply. If you have not yet created the desired submaps, you may return to this list after the maps have been created and select them at that time.

Defining a Common Stage's Applied to Properties

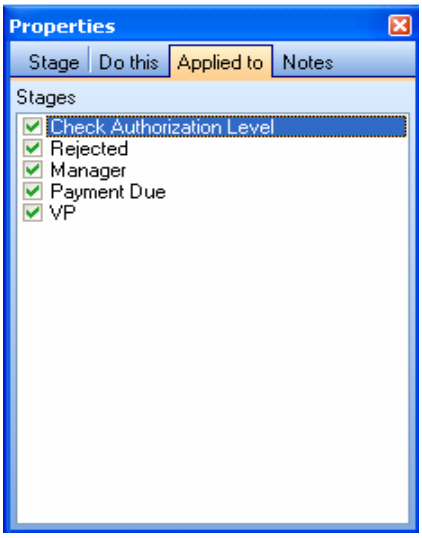


Figure 51: Applied to Tab

A Common stage is used to define actions or formulas that should be available from multiple stages in a map. An example of the use of a Common stage might be the requirement for a warning to be issued an hour after a deadline has passed, or to allow users to add comments to folders.

Use the **Applied to** tab to specify a list of stages (selected from the map's User, Group, System, and Sub-procedure stages) where any actions or formulas from this Common stage will be available.

Select the stages to apply the Common Stage to, from the list in the **Stages** property.

3.3.3 Renaming and Deleting Stages

When a stage is renamed or deleted in a procedure, the old names are preserved in the database if the procedure has previously been published. The new names are added to the database. This is expected behavior as Users' may have old stage names on their To Do or Watch lists.

Stages can be removed from the database by purging stages using the Service Manager.

Refer to the Administration Guide for more information on Purging Stages.

3.4 Actions

In addition to the stages described above, your map needs to include actions. Metastorm BPM defines an action as the necessary step to move a folder from one stage to the next. Actions provide a mechanism by which data can be entered, modified, or evaluated within the folder.

Table 14 below summarizes the types of actions, their purpose, the rules that govern their use, and the Properties editor tabs used to define their properties:

Action	Purpose	Rules	Properties editor Tabs
User	Invoked by a user, and possibly requiring the user to enter data on a form.	You can draw user actions from any stage except an archive. You can draw user actions to any stage except a common stage. User actions can be loopback actions. Any number of user actions may be drawn from the start box. If you draw more than one user action from a stage they will be shown in the client folder in the order of the priority set on the *Action* tab of the Properties editor. User actions are the only actions that can display a form to the end-user.	Action Form Roles Do this Notes Simulation
Timed	Initiated automatically at some specified interval after a specified event in the life of a folder.	You can draw timed actions from any stage except an archive. You can draw timed actions to any stage except a common stage. Timed actions can be loopback actions. Timed actions may not be drawn from the start box.	Action Time Roles (Loopback Only) Do this Notes Simulation
Flagged	Initiated by the raising of the flag named on the Flag tab of the Properties editor. The flag may be raised by an action on some other folder (not necessarily in the same map or even procedure), or by an external application (via the event manager's DCOM or command line interfaces).	You can draw flagged actions from any stage except an archive. You can draw flagged actions to any stage except a common stage. Flagged actions can be loopback actions. Multiple flagged actions may be drawn from the start box.	Action Flag Roles (Loopback Only) Do this Notes Simulation
Conditional	Initiated automatically on some data condition. Metastorm BPM tests the condition first when the folder moves into the stage, and again whenever the folder is updated by a loopback action on that stage.	You can draw conditional actions from any stage except archive or common stages. Conditional actions can be loopback actions. Conditional actions may not be drawn from the start box. If you draw more than one conditional action from a stage, they will be evaluated in the order of priority set on the *Action* tab of the Properties editor.	Action Do this Notes Simulation

 © Metastorm Inc., 2008

Action	Purpose	Rules	Properties editor Tabs
Rendezvous	Initiated automatically when those folders defined in a subprocedure stage as requiring completion reach an archive stage. The Rendezvous can be set to trigger when all or just one of those defined reach completion.	You can draw rendezvous actions only from a sub-procedure stage. You can draw rendezvous actions to any stage except a common stage. If you draw more than one rendezvous action from a stage, they will be evaluated in the order of the priority set on the *Action* tab of the Properties editor.	Action Maps Do this Notes Simulation

*Table **14**: Action Types*

A loopback action is any action that updates a folder, but does not move it on to a different stage. By default, when a loopback action is taken on a folder, Metastorm BPM does not regenerate the list of users who have the folder on their To Do or Watch lists. It simply updates the details for this folder in each list. Using the properties on the Roles tab for a loopback action, you can modify this functionality so that the To Do and Watch lists are regenerated.

While the Designer allows you easily to make modifications to a procedure at any time, you should keep in mind that any changes you make to a "live" procedure (one that is already in use in your organization), will not be retroactive. For example, if you add a Timed or Flagged action to a stage, and there are already folders sitting at that stage, these new actions will not apply to those folders. However, new folders arriving at the stage will be affected by these changes.

The Metastorm BPM Notify Gadget only displays folders with priorities between 1 and 9.

3.4.1 Working with Actions

You can draw any number of actions from any stage except an Archive stage. With the exception of Conditional, Timed, and Rendezvous actions, you can also draw them from the Start Box. You can draw an action back to the same stage (creating a *loopback* action) or to any other stage, but you cannot draw an action to the Start Box.

Adding Actions to a Map

To add an action to a map, click on the appropriate action button on the Map toolbar. Place the action on the map by drawing a line with your mouse between the two stages that the action will connect. Actions may be repositioned on the map by clicking on their icon and dragging it to a new location. As you move an action, the connecting lines between stages are redrawn.

*To add several copies of a particular action, hold down the **Shift** key before clicking on the desired Action button.*

Action names can use characters which are configured with a default locale of North American, Western European or Central European character sets. However, action names cannot include any of the following characters: Pipe (|), Period (.) Dollar Sign ($), Equal Sign (=), Back Slash (\), Quotation Mark ("), Forward Slash (/), Percent (%), Comma (,), Plus (+), Colon (:), Semi-colon (;) and Angle brackets (<>).

Deleting Actions from a Map

Actions may be deleted from a map by selecting the desired action and pressing the **Delete** key, or by using the **Edit | Cut** option.

Actions will be deleted automatically if either of the stages they connect is deleted.

Moving Actions

Actions may be moved to connect a different stage or pair of stages. For example, *stage A* → *stage B* may be changed to *stage B* → *stage A*. To move an action:

1. Selecting the desired action and right-clicking on the mouse; or
2. Select **Edit | Move** from the menu;
    - Designer displays a **Move Action** dialog box.

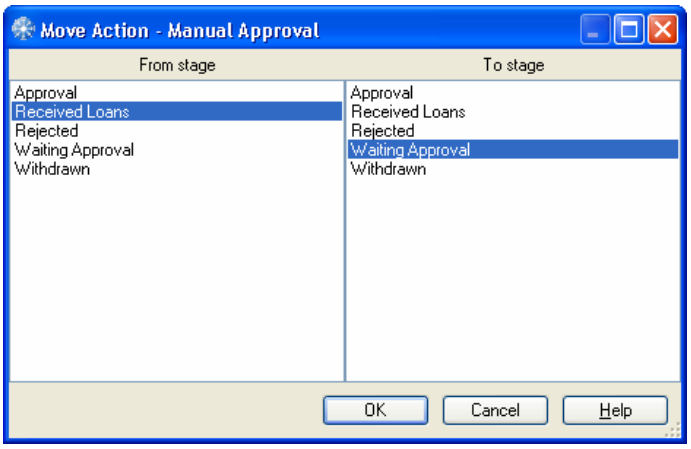


Figure 52: Move Action dialog

3. Select the star of action in the **From stage** list and next stage from the **To stage** list.
4. Click **OK**;
    - The action is repositioned.

Conditional Action Issues

Conditional actions are not triggered the instant a folder arrives at a stage. This may cause an unexpected outcome.

 © Metastorm Inc., 2008

For example, in the following map, the **Holiday Request**'s Form properties are set to **Re-open folder**.

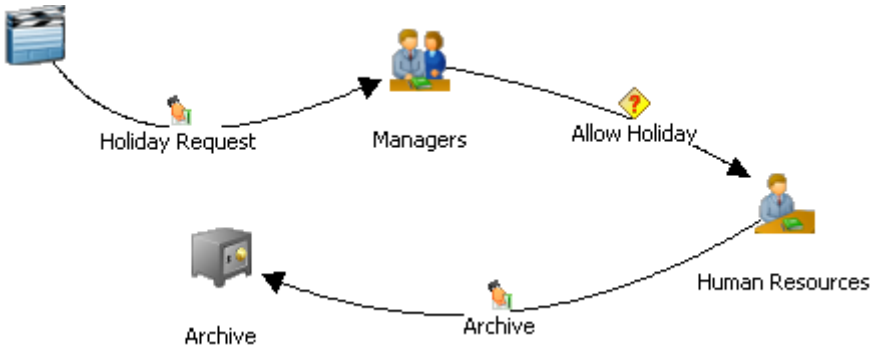


Figure 53: Example of conditional action used with Re-open folder option

When the folder arrives at the **Managers** stage (which has a conditional action leaving it) there is no guarantee at which stage (**Managers** or **Human Resources**) the folder will re-open.

The outcome could be different every time. Therefore it is recommended that you avoid this scenario.

3.4.2 Action-independent Tabs

The following action-related tabs are common to all actions:

- Action.
- Do this.
- Notes.

Defining Action *Properties*

To define an Action's properties:

1. Select the action in the Procedure Map.
2. Select the Actions tab in the Properties editor

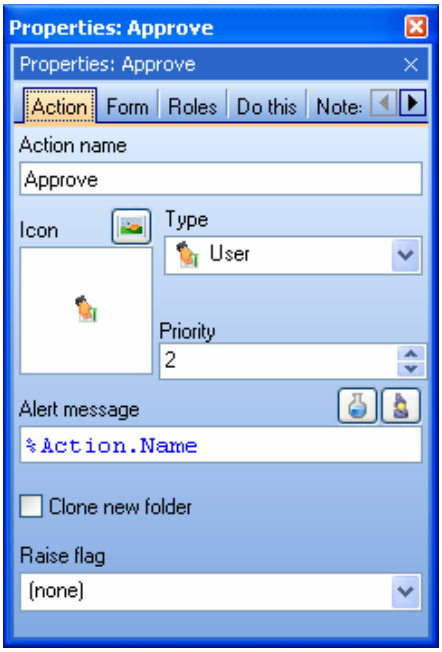


Figure 54: Action Tab

3. Enter the name for the action in the **Action name** field. The following characters are not supported in the **Action name** property:

   - Pipe (|).
   - Period (.).
   - Dollar Sign ($).
   - Equal Sign (=).
   - Back Slash (\).
   - Quotation Mark (“).
   - Forward Slash (/).
   - Percent (%).
   - Comma (,).
   - Plus (+).
   - Colon (:).
   - Semi-colon (;).
   - Open bracket (<).
   - Close bracket (>).

   For User actions, this name is displayed on the button used to invoke the action in the Metastorm client. It is also shown in the audit trail of a folder on which this action has been performed

   The maximum character length for this field is 31.

   If you do not specify a name, Designer uses the default value of Action plus an auto-incremented number.

 © Metastorm Inc., 2008

You cannot draw two actions with the same name from the same stage.

*When designing an action to expose as a web service, the **Action name** should contain only alphanumeric characters and spaces.*

4. If you wish to change the current action to a different type, choose the new type of action from the **Type** drop-down.

   If you change the action type, other properties you have specified may change. Be sure to review the properties for the new action type.

5. By default, Designer will represent the action on the map with the icon displayed in the Properties editor. To use an icon of your own choice by double-click on the icon displayed in the Properties editor, or by click on the Image button. The **File Open** dialog is displayed, and you can browse and select an alternate icon.

6. Specify the priority of the action from the **Priority** field. For user actions, the priority impacts the left-to-right order in which the action buttons appear in a folder in the Metastorm client. Priority 1 appears first, priority 2 appears second, priority 3 third, etc. The highest priority is the number of actions that leave the stage. If three actions leave a stage, you cannot have an action with a priority of 4.

   For Conditional actions, the Priority property determines which of the **Only start action if** properties is evaluated first. The Conditional action with Priority = 1 is evaluated first, followed by the Conditional action with Priority = 2, and so on.

7. When this action is completed, the **Alert message** property determines what will be displayed in the Message column for the folder in the To Do and Watch lists. By default this property contains `%Action.Name` which is a system variable representing the name of the action. You can modify this default as required. The **Alert message** property can contain a mixture of text and one or more Metastorm functions or variables. The value you enter can be typed directly into the field or you can enter it by either:

   - Integration wizard, displayed by clicking on the **Integration wizard** button: or
   - Formula editor dialog, opened by clicking on the **Formula editor** button.

   This message will also be stored in the audit trail of the folder.

8. To specify that the action should create a new copy of the selected folder and act on the copy rather than on the original, check the **Clone new folder** check box. The cloned folder will move on to the next stage leaving the original folder behind for further actions to be taken.

   The Clone new folder option is not available for an initial action.

9. If you want this action to raise one of the flags defined for this procedure select the appropriate value from the **Raise flag** drop-down.

For more information about Flags, refer to Chapter 9, Flags, on page 254.

10. When adding Action properties for a creation action, a **Description** text field is provided so that the process designer may specify a description for the form. This description will be displayed in the Blank Forms list of the Metastorm clients.

*Using the **Clone new folder** option*

The **Clone new folder** option creates a copy of the selected folder and automatically sets its parent (%parent variable) as the original folder. When a folder is cloned, the original folder remains at the same stage and the cloned folder moves to the next stage. The parent folder is not updated or changed when it is cloned (except its folder lock is removed).

All of the folder's custom variables and the following system variables are copied:

- %Category
- %Deadline
- %MapName
- %Priority
- %ServerName
- %Subject

*If a conditional action is marked **Clone new folder** and the expression is evaluated to true, a cloned folder will be created. In this case, any subsequent conditional actions will be evaluated against the parent folder in priority order.*

The audit trail (the list of actions performed on the original folder stored in the eEvent table) is not copied to the cloned folder.

Defining* Do This *Properties

The Do this tab lets you specify one or more commands to be run when the action is invoked or submitted.

 © Metastorm Inc., 2008

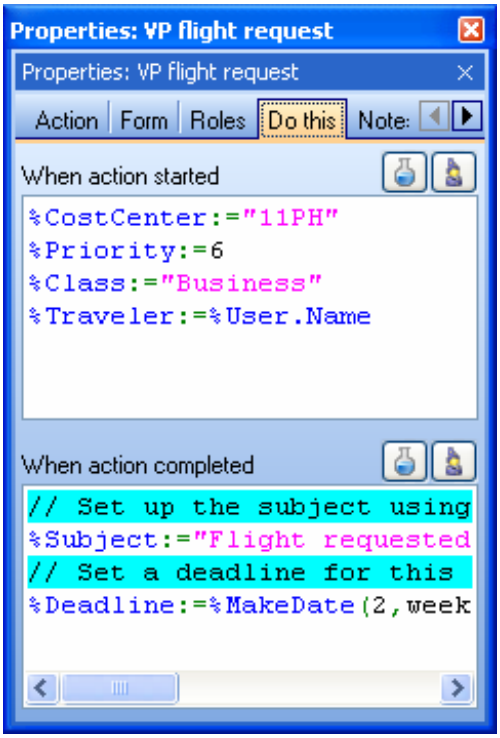


Figure 55: Do this Tab

1. For user actions only, to specify a command to be evaluated immediately after the action is started, enter it in the **When action started** property. For example, you might use this to preset form data before loading a form.

Do not use the following variables in the When action started property for the creation action:

- Folder originator (`%Originator`)
- Folder name (`%FolderName`)
- When stage started (`%EntryTime`)
- Action name (`%Action.Name`)

None of these variables have any value until after a folder has been created, or in the case of "When stage started," not until the folder has entered the first stage. If used, they will either return a blank or the form will not load at runtime.

2. For non-user actions only, to specify whether Metastorm BPM should pursue the action, enter a condition in the **Only start action if** field. The action is evaluated immediately before it is invoked, and the action start only if it evaluates to TRUE.

3. To specify processing after the action is completed, such as updating a database, printing a document, or driving an external application, enter a command in the **When action completed** field.

Defining Notes *Properties*

The **Notes** tab of the Properties editor displays a memo field which the process designer may use to make notes. These notes will not be seen by end users.

Defining Simulation Properties

The **Simulation** tab allows you to define the settings required to run a simulation of the procedure and process the results through a process analysis tool.

The Simulation tab is available for each Action in the procedure: however, the available fields differ for a creation action (the first action on the map) and subsequent actions.

Defining a Creation Action's Simulation Properties

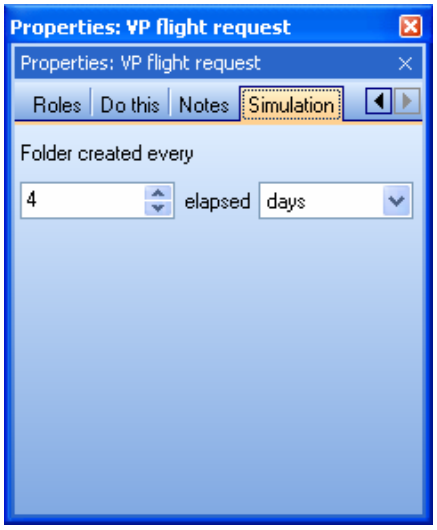


Figure 56: Simulation Tab for Creation Action

The **Simulation** tab for a creation action (i.e. the first action on a map) requires you to specify how often a folder will be created for the simulation.

1. In the **Folder created every** property, enter a number.
2. In the **elapsed** property, select the appropriate unit of time from the drop-down. You may select from:

    - minutes
    - hours
    - days (1 day = 8 hours)
    - weekdays (1 weekday = 8 hours)
    - weeks (1 week = 5 days)
    - months (1 month = 20 days)

 © Metastorm Inc., 2008

Defining Simulation *Properties for Later Actions*

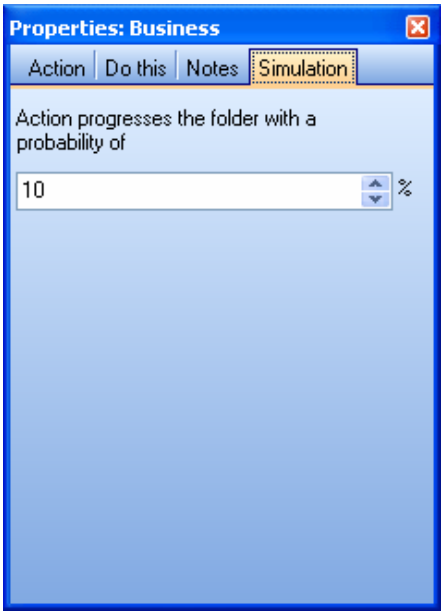


Figure 57: Simulation Tab for Later Action

The Simulation tab for normal actions requires you to specify the probability that the selected action will progress a folder.

1. In the **Action progresses the folder with a probability of** property, enter a number to provide the probability in percent.

3.4.3 Action-specific Properties

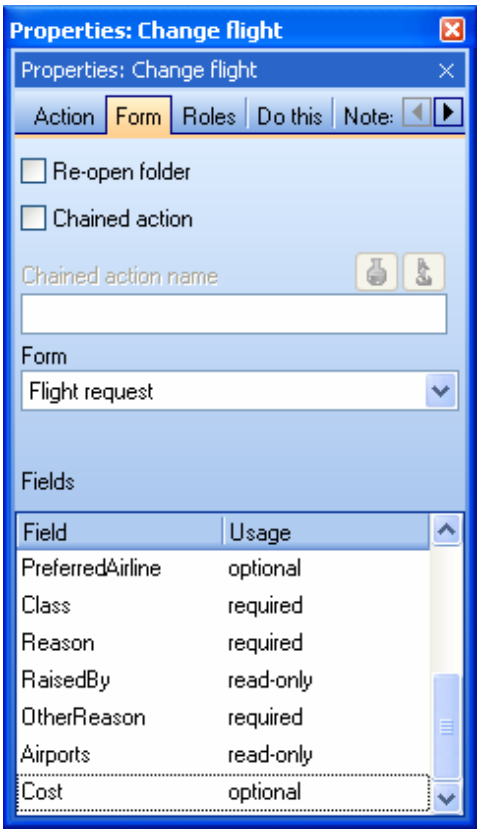


Figure 58: User Action Form Tab

Defining a User Action's Form *Properties*

The Form tab of the Properties editor lets you specify the form to be displayed when the user action is initiated.

1. When the **Re-open folder** checkbox is checked, the user is returned to the folder when the action is submitted.
2. When the **Chained action** checkbox is checked, the user is moved to the next action in the series of chained actions when the action is submitted.

If you select the **Chained action** checkbox, you must provide the name of the action that should be invoked next. You can do this by typing the action name into the **Chained action name** field or by using the Integration wizard or Formula Editor.

Chained actions allow you to break large forms into a number of separate smaller forms that are presented to the user in sequence. It also allows you to direct the user to different sections according to the values they enter.

See the Wizard sample procedure for an example of using chained actions.

The option to chain actions is not available in actions with no associated form.

 © Metastorm Inc., 2008

The options to chain actions and re-open folders are not available on actions that result in a System, Sub-procedure or Archive stage.

For security purposes, it is advisable to avoid the use of chained actions and the re-open folder option in procedures that may involve Public Access.

It is not possible to chain between actions associated with PDF forms and actions associated with HTML forms.

When chaining to an action associated with an HTML form, be aware that if the user cancels the HTML form, the original folder will remain on display.

The two checkboxes described above are mutually exclusive; i.e., they cannot both be checked. If neither box is checked, the user will be returned to the desktop when the action is submitted.

3. Select the form to be used from the **Form** drop-down. This list is generated from the forms associated with this map and any common forms associated with this procedure. If you select **(none)**, the default value, you can check the **Confirm action** checkbox to specify that a dialog is displayed requesting confirmation that the action should be committed.

If you select a form, the Properties editor will display an extra grid listing the fields on the form. Specify how each of the fields is to be displayed for the user who undertakes the action. To do this, click on the usage column for the field and use the dropdown. Depending on how the fields' properties have been defined, you may select from any of the following usage options:

- Optional field
- Required input field (the form cannot be submitted until the field has been completed— not applicable for a calculated field, a checkbox, or a button)
- Read-only field (not applicable for a button)
- Hidden field. A field will be hidden when a user is undertaking this action but not when the folder is viewed at a stage.

The default usage value for each field is defined by the process designer when the fields are added to the form.

For more information, refer to Chapter 6, Forms, page 125.

To change the default value, click in the **Usage** column for each field to see a drop-down, and make your selection from the available usage options.

*If the field **Usage** property is set to "hidden" for a particular action, when a user clicks on the action and the form is loaded, the hidden field is not sent to the browser. Therefore, no other fields on the form can reference (i.e. have visibility dependent on) the hidden field.*

Defining a User Action's Roles *Properties*

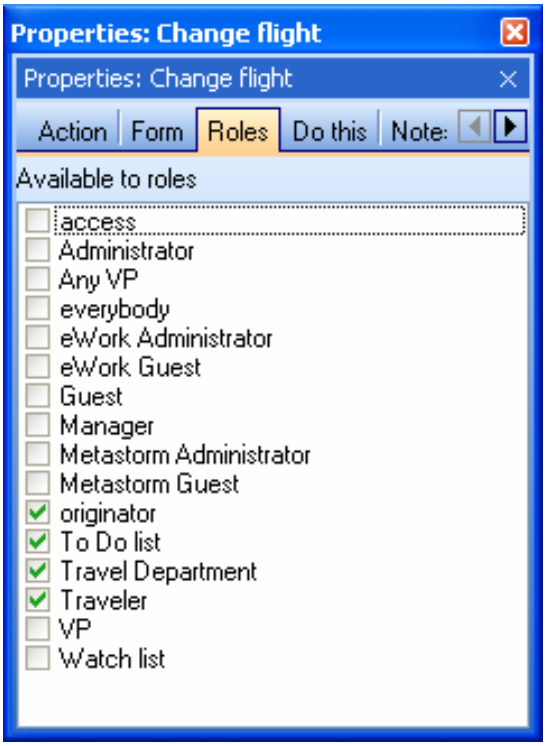


Figure 59: User Action Roles Tab

When defining a **User** action, you specify the roles allowed to perform this action. An action is available to any user who holds the roles specified in this property, no matter whether the folder is on their To Do list or Watch list. If they do not hold any of the roles selected, then that action button will not appear in the folder when they view it.

Select the Roles tab to display a list of roles defined for the procedure.

Select the roles from the **Available to roles** list. You can use the "To Do list" role to include any users who have the folder on their To Do list. This is the default value. This role is not available for an action leaving the Start Box, for which the default role is "everybody." You may select more than one role: for example "To Do List" and "Vice President." Selecting a role in this property does not cause a folder to appear on a user's To Do or Watch list. It only provides them with the authority to undertake the action if they can access the folder.

*If all the roles on the **Available to roles** list are unchecked, the action is not available to anyone.*

For further information on using the 'everybody' role, refer to section 8.1.8, Using the 'everybody' role

Defining a Loopback Action's Roles Properties

A *Loopback action* is any action that updates a folder, but does not move it on to a different stage. A loopback action updates the value in the database storing the time the folder was last updated, but not the value storing the time the folder entered the stage.

 © Metastorm Inc., 2008

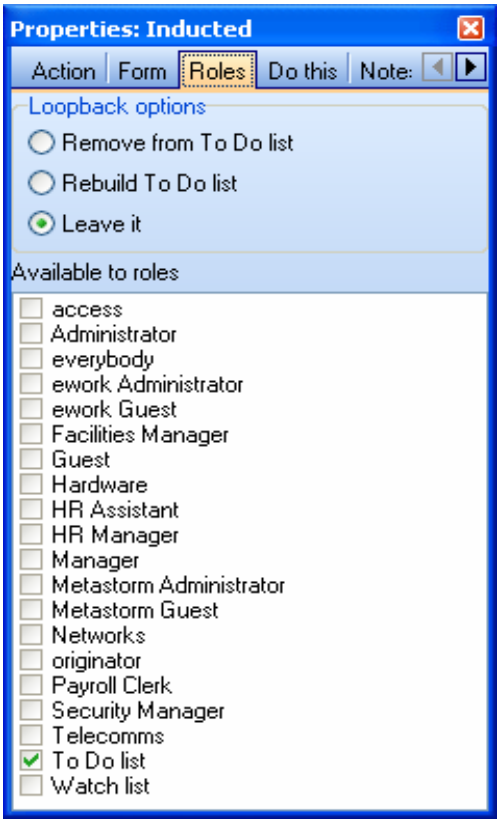


Figure 60: Loopback User Action Roles Tab

If the user action is a loopback action, a radio group of loopback options is included on the Roles tab.

Select one of the following three **Loopback options**:

- **Remove from To Do List**

  Selecting this option instructs the system to remove the folder from the *To Do* list of the user who invoked the action. This is useful in Group stages, where the folder may be initially on several users' *To Do* lists. As each user performs this action, the folder is removed from that user's *To Do* list and is updated on all the other users' lists.

  In this case, you will probably require an action, to remove the folder from the stage when there is no one left on the To Do *list. Otherwise the folder may remain at that stage, but with no action to progress it to the next stage.*

- **Rebuild To Do List**

  Selecting this option instructs the system to regenerate the list of users who have the folder on their *To Do* or *Watch* lists, just as though the folder had arrived from another stage. This is useful in a stage at a point when some update occurs that changes the data used in evaluating a dynamic role. Unless the *Rebuild To Do list* option is checked, the data change would not affect the To Do list.

- **Leave it**

  Normally, when a loopback action is taken, Metastorm BPM does not regenerate the list of users who have the folder on their *To Do* or *Watch* lists. Selecting this option instructs the system to update the folder data affected by the action but not to re-evaluate the users on the To Do and Watch lists. This is the default option.

Defining a Timed Action's Time *Properties*

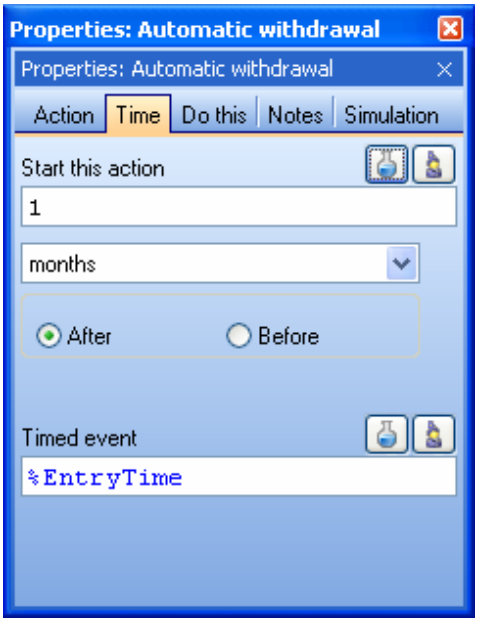


Figure 61: Timed Action Time Tab

If you are defining a Timed action, you need to identify the time when the Metastorm Process Engine will start the action. You do this by specifying the number of time units that are to elapse before the action begins. The default Time tab property settings start the action "1 hour after %EntryTime."

To set a *Timed Action's* start time:

1. Enter the number of time units in the **Start this action** field. This can be a Metastorm expression. Select the time units to be used from the drop-down in the second field. You may select from minutes, hours, days, weekdays (Monday through Friday), weeks, or months. The default setting is *hours*.
2. Specify whether the action is to be invoked **Before** or **After** the event by selecting the appropriate radio button.
3. Use a variable or formula to specify the event before or after which the action should be initiated in the **Timed event** property. Select from the available options presented through the Integration wizard**,** or enter the variable or expression directly through the Formula editor.

 © Metastorm Inc., 2008

Defining a Timed Action's Roles Properties

The **Roles** tab is available only when the timed action is a loopback action. The **Roles** tab allows you to select one of two options available for a loopback Timed action, **Rebuild To Do list** or **Leave it.**

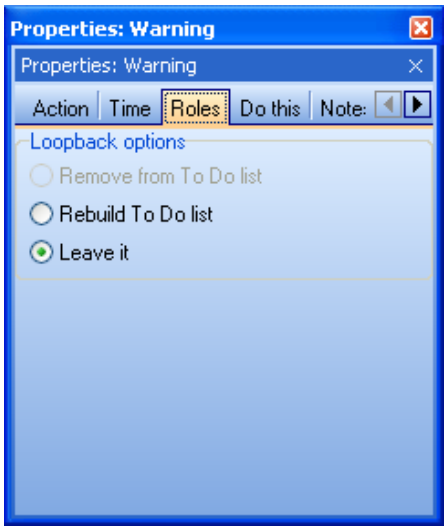


Figure 62: Timed Action Roles Tab

- **Rebuild To Do list**

  Selecting this option instructs the system to regenerate the list of users who have the folder on their *To Do* or *Watch* lists, just as though the folder had arrived from another stage. This is useful in a stage at when some update occurs that changes the data used in evaluating a dynamic role. Unless the *Rebuild To Do list* option is checked, the data change would not affect the To Do list.

- **Leave it**

  Normally, when a loopback action is taken, list of users who have the folder on their *To Do* or *Watch* lists is not regenerated. Selecting this option instructs the system to update the folder data affected by the action but not to re-evaluate the users on the To Do and Watch lists. This is the default option.

*The **Remove from To Do list** is not a valid option for a loopback Timed action and is inactive.*

Defining a Flagged Action's Flag *Properties*

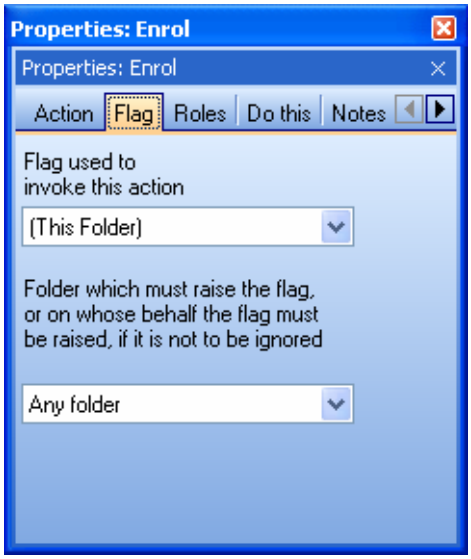


Figure 63: Flagged Action Flag Tab

Select the flag that is to initiate the Flagged action from the list in the **Flag used to invoke this action** field. This is a list of all flags defined for the procedure. The default value is blank (nothing selected).

You can define the Flagged action so that it only acts on folders with the specified relationship to the folder that raised the flag. This relationship is established in the **Folder which must raise the flag…** property. The options are:

- **Any folder**

  Use this option when the action should be invoked no matter which folder raises the flag. This option should *always* be used for an action that creates a new folder.

- **This folder**

  Use this option to respond to a flag raised by an external application, via the *eRaiseFlag* interface specifying *FolderId* as the *FlagRaiser*. The flagged action will only start if the Folder ID in FlagRaiser matches the FolderID of the folder upon which the flag is acting.

- **This folder's parent**

  Use this option in the case of a sub-procedure, when the action should be invoked only as a result of a flag raised by the "parent" folder.

*While you can use the **Only start action if** property (on the* Do this *tab) to check whether a flag was raised by, or (through the external interface) on behalf of, a particular folder, it is much more efficient to select either the* This folder *or* This folder's parent *options. If you select* Any folder*, the Metastorm Engine will inspect every folder in the stage from which this flagged action is drawn, to determine whether its **Only start action if** formula evaluates to* TRUE*. By using* This folder *or* This folder's parent*, the Engine will check only folders awaiting flags from this specific folder.*

Defining a Flagged Action's Roles *Properties*

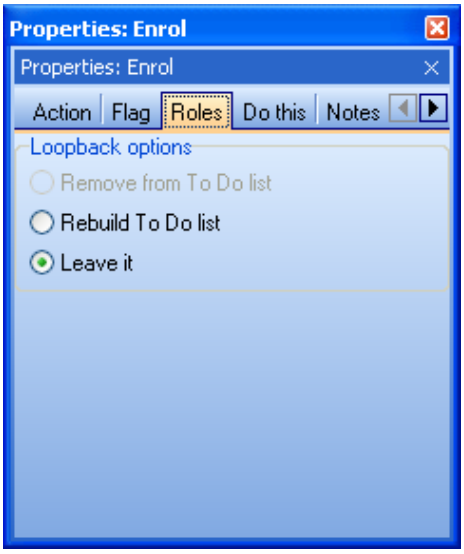


Figure 64: Flagged Action Roles Tab

The Roles tab is available only when the Flagged action is a loopback action. The Roles tab allows you to select one of two options available for a loopback Flagged action:

- **Rebuild To Do list**

  Selecting this option instructs the system to regenerate the list of users who have the folder on their *To Do* or *Watch* lists, just as though the folder had arrived from another stage. This is useful in a stage at a point when some update occurs that changes the data used in evaluating a dynamic role. Unless the *Rebuild To Do list* option is checked, the data change will not affect the To Do list.

- **Leave it**

  Normally, when a loopback action is taken, Metastorm BPM does not regenerate the list of users who have the folder on their *To Do* or *Watch* lists. Selecting this option instructs the system to update the folder data affected by the action but not to re-evaluate the users on the To Do and Watch lists. This is the default option.

The Remove from To Do list *is not a valid option for a loopback Flagged action and is therefore inactive.*

Defining a Rendezvous Action's Maps *Properties*

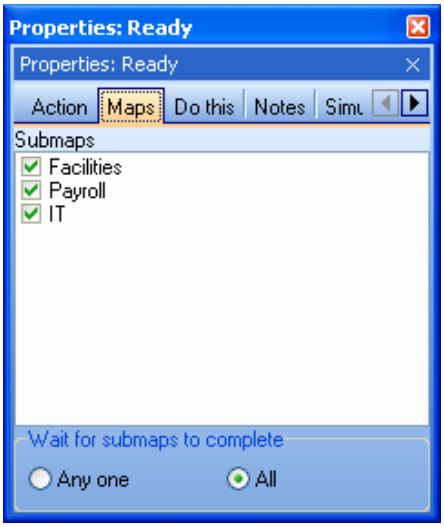


Figure 65: Rendezvous Action Maps Tab

Rendezvous actions are used in conjunction with Sub-procedure stages to move the parent folder from a sub-procedure stage when one or more of its child folders have arrived at archive stages.

When defining a Rendezvous action you need to specify the submaps associated with this action.

Select the submaps using the check boxes in the **Submaps** property.

In the **Wait for submaps to complete** property, click the appropriate radio button to specify:

- That the completion of **Any one** submap is adequate to initiate the rendezvous action.

OR

- That the folder must reach an archive stage in **All** the selected submaps before the rendezvous action is initiated.

Although the rendezvous action in the parent map will not be initiated until the submaps criteria is satisfied, other actions in the parent map can act on the folder including moving it from the sub-procedure stage.

 © Metastorm Inc., 2008

4 CUSTOM AND SYSTEM VARIABLES

A variable is a container of information. When defined in the Metastorm database, a variable can be called upon when you customize your Metastorm processes. There are two types of variables: custom and system.

Custom variables are user-defined, and are the fields of the custom variable table created in the database when a procedure is published. One custom variable table is created for each map in the procedure. The table has the same name as the map, with any spaces replaced by underscores.

Any limitations that apply to the database platform on which the Metastorm database is running, apply to custom variable tables. This includes constraints such as field name length (i.e., custom variable name length), number of fields per table (and therefore the number of custom variables per map) and data format. When procedures are published memo and text field content are stored as Unicode and the format of the Unicode is defined by the database.

Some variables will be common to most procedures, (e.g., creation time, folder creator, deadline). To prevent duplication Designer has predefined S*ystem variables* which can be used in any procedure. A full list of system variables is provided later in this section.

The most commonly used expression in Designer is the Assign value function. This expression assigns a value to a particular system or custom variable. For example, the `%Subject` variable could be set to a text value, or the `%Deadline` variable could be set to a calculated expression. The basic syntax for this function is:

```
%[Variable] := [Value]
```

4.1 Custom Variables

Designer automatically establishes a number of variables that can be used in calculations or in reporting. In addition, when designing a procedure you can define your own custom variables and assign values to them.

Custom variables are used to store data that has been entered by either a user or extracted from an external system. Custom variables are associated with a particular map, and the data from the

custom variable is stored in the map's database table. The custom variables form the columns of the table while the folders make up the rows.

4.1.1 Custom Variables and Forms

Typically, a form will contain one or more fields that are used to display and capture data. These fields may be associated with pre-defined system variables, or custom variables that you specify or are calculated. Custom variables are stored in the Metastorm database.

Each map in a procedure is associated with a database table. Each custom variable has its own column in this table. Before associating custom variables with fields on a form, the form itself must be linked to the particular map in whose table these custom variables are to be stored. The maximum number of variables per map depends on the maximum number of columns allowed in a table in the database being used.

Forms using custom variables must be associated with a particular map. Each map in a procedure is shown in the Procedure Explorer. Forms are associated with the map in which they appear in the Procedure Explorer. To associate a form with another map in the procedure, drag-and-drop the form icon to the new map.

If you move a form to a new map, you will need to re-create any variables used on the form and associate them with the new map.

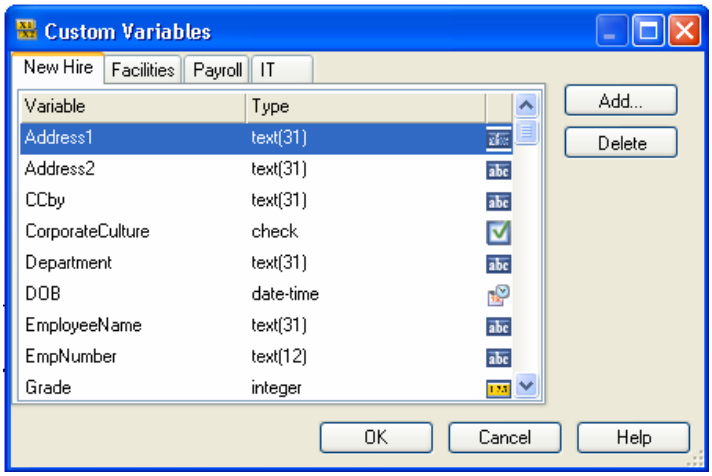
Also keep in mind that you should review any forms associated with a particular map before deleting map variables.

Forms using only system variables may be used with any map in the procedure, and may be placed under the Common item or under any map item within the Procedure Explorer.

4.1.2 Creating a Custom Variable

To create a new custom variable:

1. Click on the custom variable button, or select **View | Custom Variables**
   - Designer will display existing custom variables.



 © Metastorm Inc., 2008

Figure 66: Custom Variables Dialog

2. Click on the **Add** button to display the **Custom Variable** dialog:

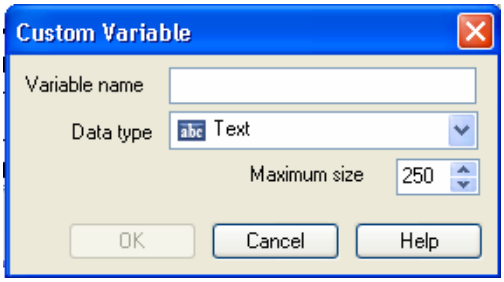


Figure 67: Custom Variable dialog box

3. Enter a name for the variable in the **Variable name** field. The variable name:

   - Must be unique for each new custom variable associated with this map.
   - Must be no longer than:
   - 31 characters if you are using SQL Server
   - 30 characters if you are using Oracle.
   - Must be comprised of only alphanumeric characters, without spaces.
   - Must not begin with a number
   - Should not begin with the letter 'e'. This will help avoid scripting errors.
   - Must not include underscores if the variable is to be exposed as a web service.
   - Must not include reserved words.

   A number of words have special meanings in the databases that Metastorm BPM supports. You should not use any of these reserved words when naming a variable. Refer to the `Reserved Words.txt` *file for a list of reserved words. This file is located in the Metastorm BPM\Designer folder.*

   - Must not include system variables. A list of system variables is contained within the oemdesigner.xml file under the <Variables></Variables> node. This file is located in the Metastorm BPM\Designer folder.

4. Select the **Data type**, as follows:

   - **Integer** - whole numbers (no decimal places). The maximum number of digits for an integer field is 10. If you want to enable the user to enter a number with more than 10 digits, use a Real number.
   - **Check** - for the entry of True or False values only.
   - **Memo** - for the storage of large amounts of text.
   - **Text** - standard (alphanumeric) text of up to 250 characters.
   - **Real** - numbers up to 15 decimal places.
   - **Date-time** - dates and times.
   - **Currency** – for entry of currency formats

   Metastorm BPM automatically sets the default data type to match the field type you are defining.

You can change the data type, but if you do, the new variable you are defining cannot be used for the data field you are defining. (The only exception is for a Number *field, where you may choose either a* Real *or an* Integer *variable.) Once you have created the variable, you cannot change the data type.*

The Memo data type is used for complex types. For further information refer to 5 Complex Types.

5. If you are defining a **Text** variable, you also need to enter a value in the **Maximum size** field. This field is the size used to hold the name of any Metastorm object, such as a folder, server or user. The default size for text fields, radio group and drop-down fields is 250 characters. If you need to provide more room or allow for multiple lines of text, you should use a memo field and memo variable.

Metastorm BPM creates the variable as a column in the database table for the map with which this form is associated.

Custom variables must be associated with a specific Metastorm map, and are available to folders associated with that map; however, it is possible to use Metastorm functions (such as flags and SQL queries) to make custom data from folders in one map available to folders in another map.

4.2 System Variables

The following Metastorm system variables are defined:

- Action and Session variables.
- Folder variables.

Action and session variables are used to hold data specific to the Metastorm Process Engine; to the currently executing action; or to the requesting user. Unless indicated otherwise, they are read-only. The following table lists available system variables:

Variable Name	***Variable Type (size)***	***Parameters***
%Action.Name	*text (31)*	The name of the current action, as defined in the procedure map.
%Action.Notes	*memo*	The memo (if any) to be written to the folder's audit trail when the action is committed. This variable is read-write in an action. The result may be read (in a folder or action form) using the (read-only) **%Notes [<event number>]** variable. This variable is available only in an action form.
%Action.StartsStage	*text (31)*	The name of the stage that the action will move the folder to. This is normally the target stage of the action, as drawn in the procedure map. If the action is a loop-back, then this variable will hold an empty string. If the target is a system stage, with its **Auto-forward folder** to property set, then this variable will hold the name of the stage to which the folder will be auto-forwarded. This variable is available only in an action form.
%Form.Name	*Text (31)*	
%Procedure.Name	*text (31)*	The name of the current procedure.
%Procedure.Version	*text (31)*	The version number of the current procedure.
%Session.FlagName	*text (31)*	The name of the flag which triggered the action. This variable is available only in a flagged action.

 © Metastorm Inc., 2008

Variable Name	*Variable Type (size)*	*Parameters*
%Session.FlagData [*column number*]	*memo*	The contents of the specified column of flag data. This variable is available only in a flagged action.
%Session.FlagRaiser	*text (31)*	The Folder ID of the folder which raised the flag, or (via the Engine's automation interface) for which the flag was raised. This variable is available only in a flagged action.
%System.Name	*text (31)*	The name of the Engine evaluating the expression.
%System.Roles	*memo*	The list of currently published roles on this Engine's database.
%System.Time	*date-time*	The date and time when the formula is evaluated by the Engine.
%System.Users	*memo*	The list of currently registered users on this Engine's database.
%User.Error	*text(31)*	A custom error message specified by the process developer. This variable is only available in an action and will be reset after committing or canceling that action. Setting this variable causes the commit to fail and the error message to be given to the user.
%User.Form	*text (31)*	The name of the current form. This variable is available only in a form.
%User.Input	*memo*	The value just entered by the user into a data field and not yet validated. This variable is available only during the evaluation of a field's **When pressed, When changed,** or **When user selects row** property.
%User.Input [*column number*]	*memo*	The value of the specified column of the currently selected grid row. This will be empty, except during the evaluation of the grid's **When user selects row** property.
%User.Name	*text (31)*	The user's registered Metastorm user name. This will be empty during a non-user action.

Table 15: System Variables

4.3 Folder Variables

Folder variables are used to hold data specific to the currently selected folder. Unless indicated otherwise, they are read-write within an action or action form, and read-only elsewhere. The following table lists available folder variables:

Variable Name	*Variable Type (size)*	*Parameters*
%ActionCount	*number*	The number of actions committed on this folder to date. This will be zero during a creation action, and one when the creation action has been committed. This variable is always read-only.
%Archived	*Boolean*	Indication of whether a folder is archived. Returns `True` if the folder is or has ever been at an Archive stage, else returns `False`. This variable is always read-only.
%Category	*text (31)*	Used to categorize folders, under the control of the designer.

Variable Name	Variable Type (size)	Parameters
%CreationTime	*date-time*	The date and time at which the folder's creation action was committed. This variable is always read-only.
%Deadline	*date-time*	Used to control timed actions that are based on the folder's deadline.
%EntryTime	*date-time*	The date and time at which the folder entered its current stage. This variable is always read-only.
%FolderID	*text (31)*	The unique internal identifier for the folder. This variable is always read-only.
%FolderName	*text (31)*	A user-readable identifier for the folder (which may or may not be unique). This variable is preset when the folder is created, from the map's Folder prefix and Length of numeric suffix properties, but may be changed during an action.
%MapName	*text (31)*	The name of the map within which the folder was created. This variable is always read-only.
%Message [*event number*]	*text (250)*	The alert message generated for the specified event number for this folder. The folder's creation is event number zero. This variable is always read-only.
%Notes [*event number*]	*Memo*	The value of %Action.Notes stored for the specified event number for this folder. The folder's creation is event number zero. This variable is always read-only.
%Originator	*text (31)*	The registered Metastorm user who created this folder. This will be empty for a folder created by means of a flagged action. This variable is always read-only.
%Parent	*text (31)*	The Folder ID of the folder's parent. This is preset automatically for folders created by means of a subprocedure, and for folders cloned from another folder, but may be changed during an action.
%Priority	*Number*	The current priority (from one to nine) of the folder.
%ServerName	*text (31)*	The Engine server which (currently) holds the master copy of the folder. This variable is always read-only.
%StageName	*text (31)*	The folder's current stage. This will be empty during a creation action, and will identify the source stage of any other action. This variable is always read-only.
%Subject	*text (250)*	The subject for this folder. This variable is preset from the map's Subject property when the folder is created but may be changed during an action.
%Updated	*date-time*	The date and time at which the folder's last action was committed. This will be unspecified during a creation action. This variable is always read-only.

Table 16: Folder Variables

The Metastorm BPM Notify Gadget only displays folders with priorities between 1 and 9.

4.4 Assigning a Value

The Assign Value function is the most commonly used expression in the Integration wizard. It is used to assign a value to either a system or custom variable. To assign a value to a variable:

1. Select the variable in Procedure Explorer.
2. In the Properties display click the Integration Wizard icon:
    - The Integration Wizard – Subject list is displayed.

 © Metastorm Inc., 2008

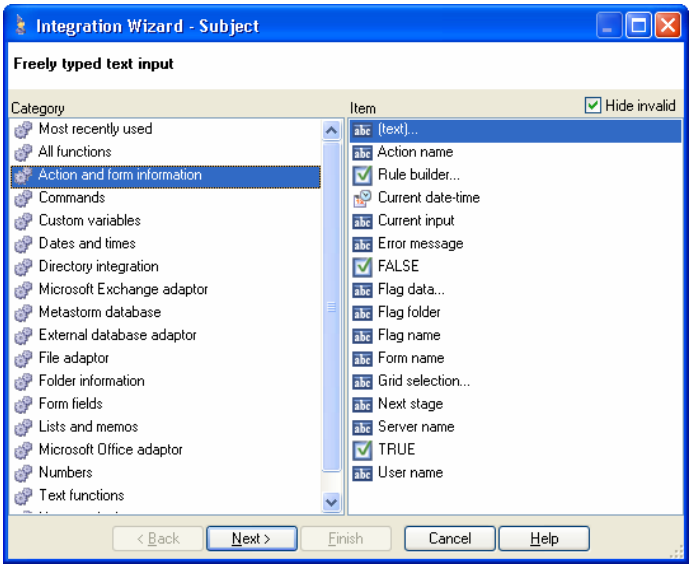


Figure 68: Integration Wizard - Subject list

3. Select a Category in the left column;
    - The available items for that are displayed in the right column.
4. Click the **Next** button;
    - An entry box is displayed;

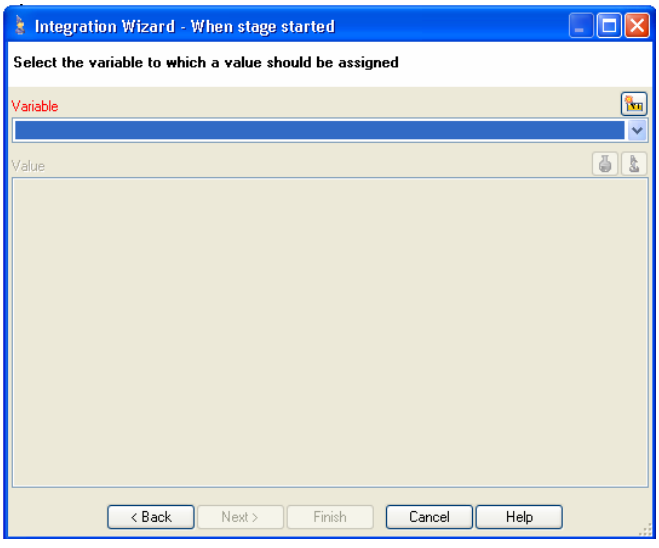


Figure 69: Assigning a value

5. . Enter details of the variable using the sequences dictated by the wizard

Alternatively the **Formula editor** button, next to the Wizard icon in the Properties display will call the formula editor to enable you to write an expression independently.

When the Value expression has been entered, click the **Finish** button.

5 COMPLEX TYPES

Complex Types are data types which can contain simple data types and / or one or more nested complex data types. The Designer has been extended to provide support for Complex Data Types.

To use Complex Types, the Metastorm Process Activators (which ship with the Process Orchestrators) interrogate .NET Assemblies / Web Services / Enterprise JavaBeans and create Integration Wizard Collection Libraries which contain exposed methods as Integration Wizard functions. Associating a library with a procedure makes the exposed methods available as functions via the Integration Wizard. When these functions use complex type parameters, the Process Designer maps complex types to custom variables using a graphical interface.

In addition to the generated library, Metastorm BPM provides the `ComplexTypesUtils.xel` support library.

This section discusses:

- Generating a Complex Types Library
- Retrieving Data from Complex Types
- Retrieving data into fields
- Populating a List box
- Populating a Dropdown

Map Complex Type Dialog

- ComplexTypesUtils Support Library.

5.1.1 Generating a Complex Types Library

Before Complex Types can be mapped a Complex Types library is generated using a Process Orchestrator Activator.

Refer to the Process Orchestrator documentation for details on using the Activators.

 © Metastorm Inc., 2008

1. Run an activator and activate an assembly, web service or Enterprise JavaBeans as applicable.
2. A Metastorm Integration Wizard Collection library is generated containing the activated methods.
3. In the Designer, publish the generated Metastorm library.
4. Associate the published library with a procedure.
5. Use the Integration Wizard to invoke the activated methods.

5.1.2 Retrieving Data from Complex Types

To retrieve data from a complex type, a memo custom variable is setup to store the data retrieved. The data retrieved can be output to fields on a form, for example text and numeric fields, or serialized as XML, Binary or SOAP. Binary is the default serialization. The example below retrieves complex data in text and integer fields which is displayed in the browser.

5.1.3 Retrieving data into fields

This example invokes a method which returns a complex type. Elements of the complex type are mapped to Metastorm custom variables.

To map complex type simple data type members to Metastorm custom variables:

The steps below assume an Activator has already been run and the resulting Integration Wizard Collection library has been associated with the current procedure.

1. Create custom variables for the input and output parameters.
2. Add fields to a form.
3. Assign custom variables to all fields.

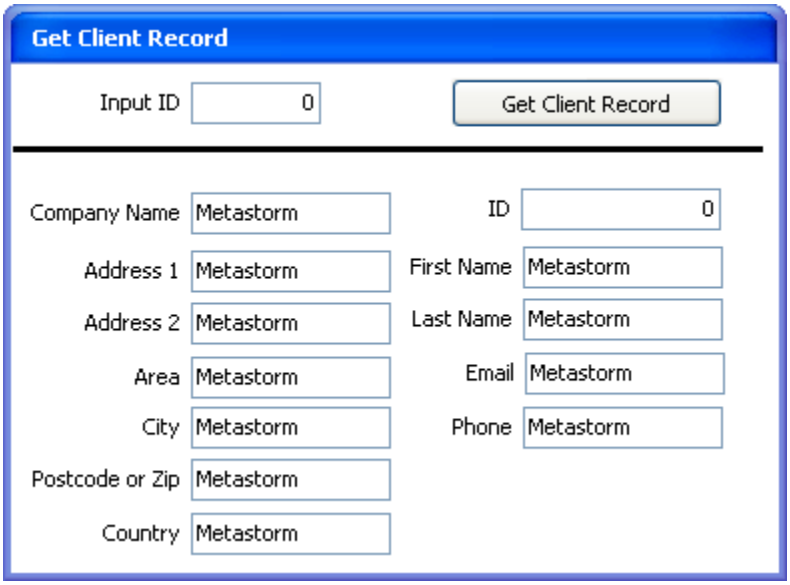


Figure 70: Complex Types Form Example

4. Add a memo custom variable to the Map. This custom variable will be used to store a serialized representation of the Complex Type data.

This memo custom variable can serialize the complex types data as XML, SOAP or Binary.

5. In the **Do This** for the Output fields set **Field is dependent on another**.
6. Select the event to use for accessing the web service (such as the 'Get Client Record' button in the example above).
7. In the **Do This** tab click the **Integration Wizard** button.
8. Select **Assign a value**.
9. Select the memo custom variable.
10. Click on the **Integration Wizard** button for the **Value**.
11. Select the generated library and the method you wish to invoke which returns a complex type.

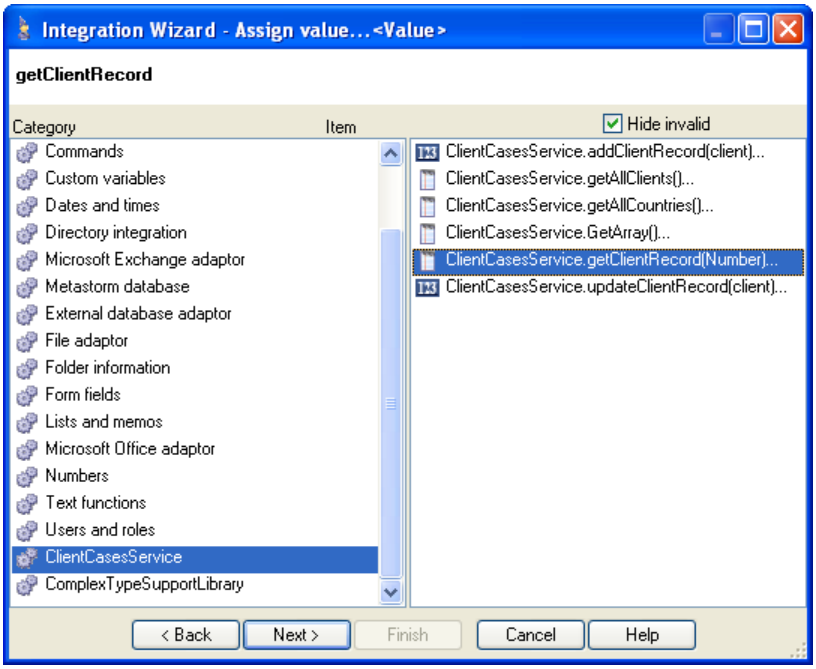


Figure 71: Web Service Example

12. Click **Next**.

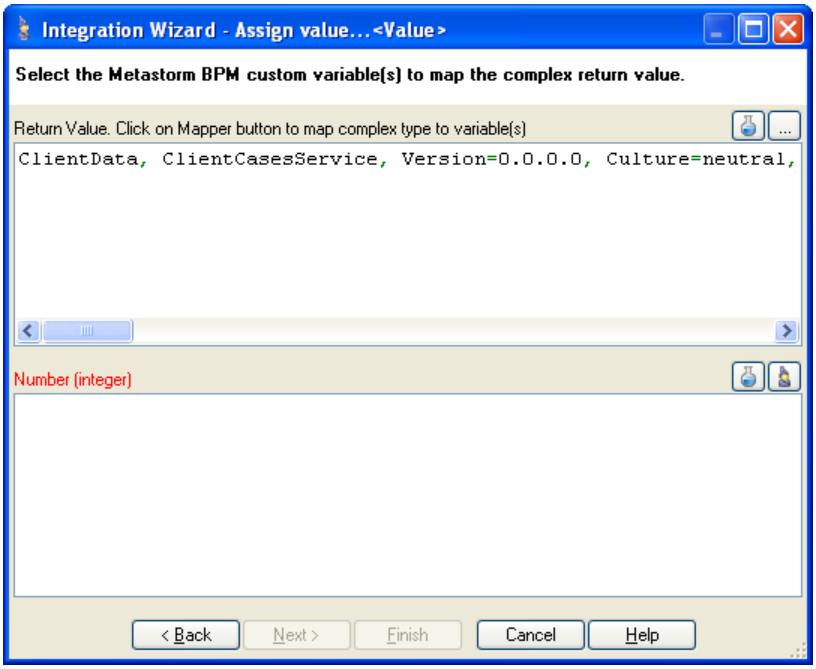

Integration Wizard - Assign value...<Value>
Select the Metastorm BPM custom variable(s) to map the complex return value.
Return Value. Click on Mapper button to map complex type to variable(s)
ClientData, ClientCasesService, Version=0.0.0.0, Culture=neutral,
Number (integer)
< Back
Next >
Finish
Cancel
Help

Figure 72: Complex Return Value Example

13. Click the Mapper button to map the complex type returned by the invoked method to custom variables.
14. Map the complex type simple data type member to a Metastorm custom variable.

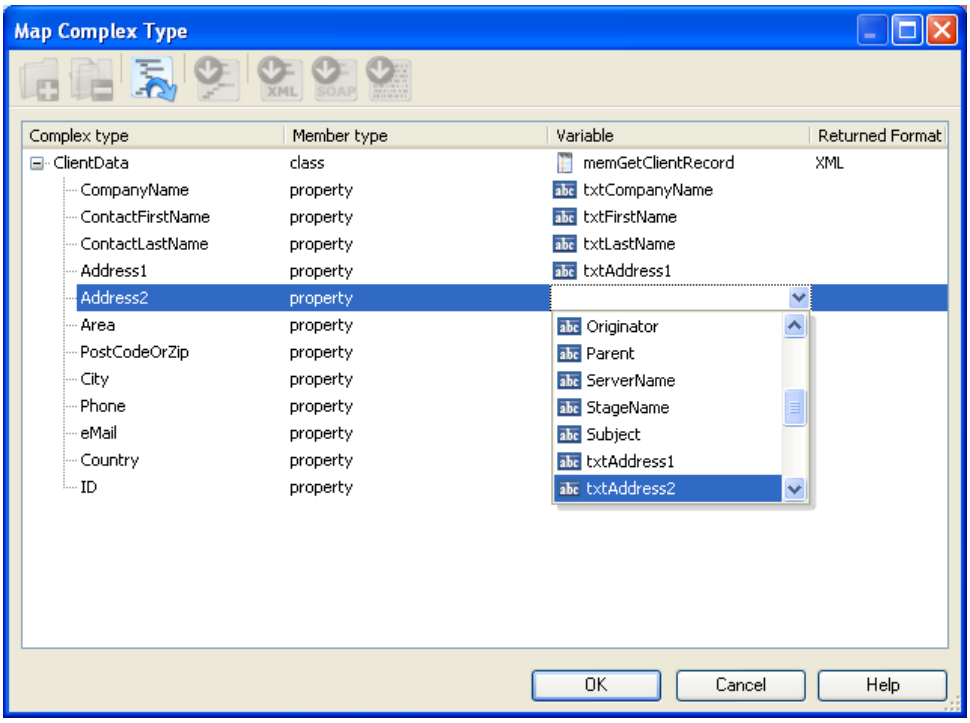


Figure 73: Mapping Complex Types

In order for the complex type to return XML or SOAP serialization, select a memo custom variable, click the return value button and assign XML or SOAP serialization. The default return serialization is Binary. Refer to section 5.2.4 Serialization for more information.

15. Click **OK**

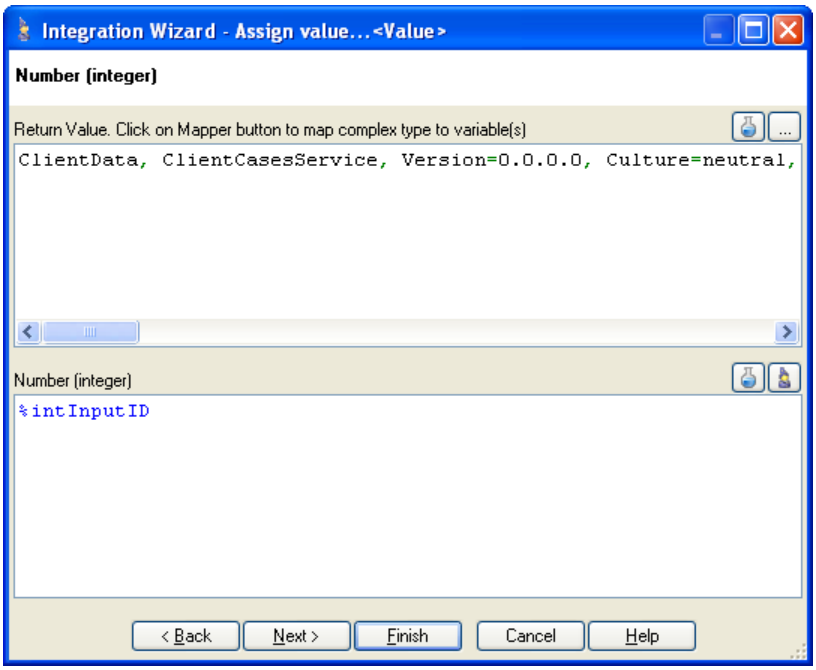


Figure 74: Input Field custom variable

16. Assign an input field.
17. Click **Finish**.
18. Publish the procedure.

5.1.4 Populating a List box

A list box is populated using two memo custom variables and the **ComplexTypesUtils** library function **GetList**.

If the **ComplexTypesUtils** function **GetListDirect** is used, only one memo custom variable is required.

5.1.5 Populating a Dropdown

A drop-down can only contain one field from a complex type.

Currently, only a single field/property is supported

A drop-down is populated by using two memo custom variables and the **ComplexTypesUtils** library function **GetList**. The first custom variable retrieves the data and the second specifies the exact field. The drop-down is then mapped to the second memo custom variable in List Options properties.

 © Metastorm Inc., 2008

If the **ComplexTypesUtils** function **GetListDirect** is used, only one memo custom variable is required.

5.2 Map Complex Type Dialog

The Map Complex Type dialog is used for mapping complex data type members and complex types to custom variables.

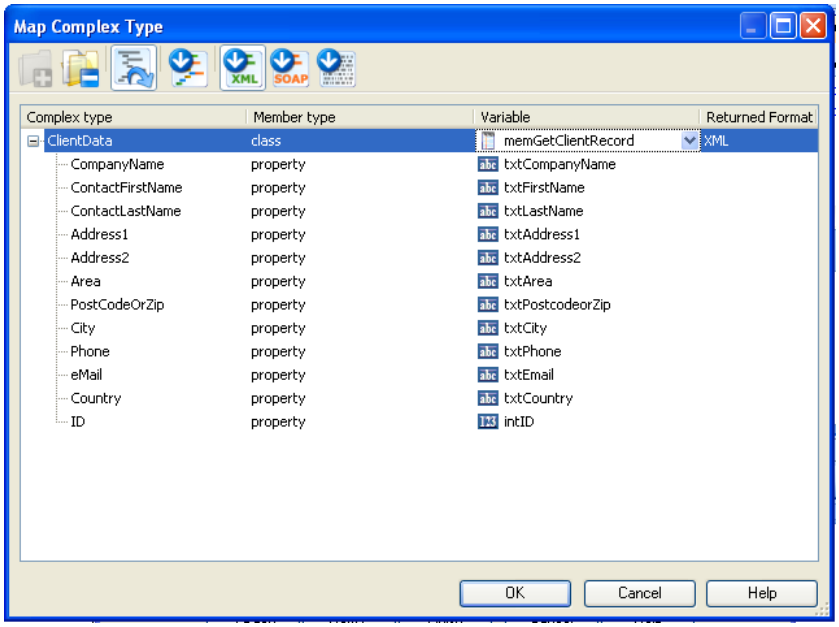


Figure 75: Complex Type Dialog

When mapping a complex type to a Metastorm custom variable the following rules must be observed:

- A complex type (the class or structure) can only be mapped to a Metastorm memo variable. If the complex type is an input parameter or reference parameter in a .NET method, the process designer must ensure that the mapped memo variable contains the serialized object at runtime. This can be achieved either through JScript.NET scripting or as a result of making another activated .NET method call. On the other hand, if the complex type is an output parameter or reference parameter in a .NET method, the Metastorm engine will serialize the .NET object to the specified memo variable using the specified serialization format. If no serialization format is specified, the default Binary format will be used.
- A complex type's public fields or public properties can be mapped to Metastorm custom variables. Simple .NET data types can be mapped to the appropriate Metastorm custom variable. For complex data types the previous bullet point applies. If the complex type is an input parameter or reference parameter in a .NET method, the Metastorm engine will create the object at runtime using .NET reflection. The public fields and public properties will then be updated using their mapped Metastorm custom variables. On the other hand, if the complex type is an output parameter or referenced parameter in a .NET methods,

the mapped Metastorm custom variables will be updated at the end of the method call using the object mapped fields and properties.

*The Complex Type Mapper reads custom variables from the database specified by the Engine DB registry key. If you wish to read custom variables from a different database, you must manually edit this registry key. (For an SQL database, **HKEY_LOCAL_MACHINE\SOFTWARE\Metastorm\e-work\Engine\Database Connectors\SQL Server DBC\Connection**. For an Oracle database, **HKEY_LOCAL_MACHINE\SOFTWARE\Metastorm\e-work\Engine\Database Connectors\Oracle DBC\Connection**.)*

Refer to Appendix A of the Metastorm Process Orchestrator for .NET Designer's Guide for a definition of simple .NET data types.

5.2.1 Columns

There are four columns in the Map Complex Type

- Complex Type – the name of the complex type, member and properties
- Member Type – displays whether the member is a class, struct, property or member.
- Variable – select from a dropdown the Metastorm variable to assign to the member.
- Returned Format – displays the returned serialization format. To select a serialization format, click on the Return Value button and select the required Serialization type using the serialization toolbar buttons.

5.2.2 Toolbar

The Map Complex Type dialog toolbar enables the user to expand and collapse complex type methods, add and remove return values and set the type of serialization to use.

Figure 76: Map Complex Type Toolbar

Button	Description
Expand All	Expand all members of the current tree list node.
Collapse All	Collapse all members of the current tree list node.
Return Value	Apply return value to the selected complex type or simple data type. When this option is selected, the Returned Format column displays "Default". "Default" represents the Default Serialization type. When a return value is applied the Serialization buttons become available.
Default Serialization	This button resets the serialization format of the tree list underneath the cursor back to default. By default the serialization is Binary.

 © Metastorm Inc., 2008

Serialize to XML	Serialize to XML. When the Return Value is selected, this button becomes available. When this option is selected, the Returned Format column displays "XML".
Serialize to SOAP	Serialize to SOAP. When the Return Value is selected, this button becomes available. When this option is selected, the Returned Format column displays "SOAP".
Serialize to Binary	Serialize to Binary. When the Return Value is selected, this button becomes available. When this option is selected, the Returned Format column displays "Binary".

Use one of the serialize buttons to override default serialization.

5.2.3 Return Value

The return value of an Integration Wizard function (%ScriptEval) is not necessarily the same as the return value of the external function being called. The Integration Wizard add-in allows the Integration Wizard function's return value to be a subset of any of the complex types returned by the external function call. This is important because it reduces the number of Metastorm custom variables required to manipulate complex data.

With complex types, the return value of the activated function called is normally a complex type object which is assigned to a custom variable. After the call is made, the variable will contain the value of an object serialized in Binary, SOAP or XML.

Some complex types have inner complex types. For example, suppose that we have a class Person, which has two inner classes – PersonalInfo (containing information such as name and age) and CompanyInfo (containing information such as company name and company phone). Also, suppose that we have an activated method GetPerson() which returns an object of type Person. Usually, the user will map the result from the call to GetPerson() to a memo variable, and at runtime this memo variable will be filled with the value of the serialized Person object. However, if the user selects CompanyInfo (an inner member of PersonalInfo) as a return value, then at runtime the memo variable will be filled with the value of the serialized CompanyInfo object.

5.2.4 Serialization

Serialization for Complex Types enables an object to be saved in one of the following output formats:

- XML
- SOAP
- Binary

The default serialization output format is Binary.

When a complex type is serialized, it contains additional information to indicate that the class type is in serialized form.

A Metastorm library which has been generated using the .NET Activator may return an error when one of the activated functions returns a hashtable. An error only occurs if the default serialization (Binary) is used when mapping a request, for example, with functions such as GetNameValueList.

Serializing

To serialize complex types to a custom variable:

1. Click Return Value . The selected data type is denoted with ”Default” in the Returned Format column.

2. Click a Serialization button:

   a. Serialize to XML.

   b. Serialize to SOAP.

   c. Serialize to Binary.

Right Mouse Menu

1. Right click a data type and select required options.

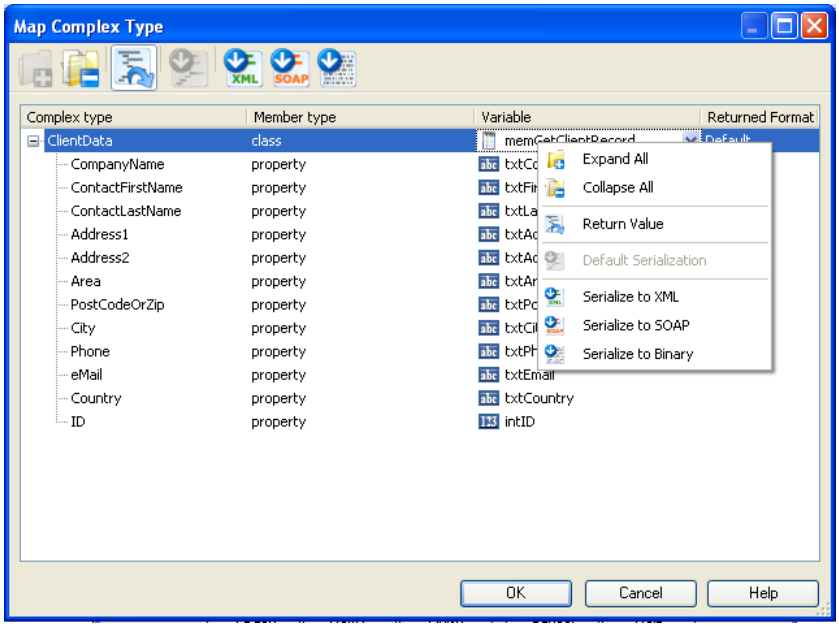


Figure 77: Serialize Complex Types Menu

 © Metastorm Inc., 2008

5.3 ComplexTypesUtils Support Library

The complex types support library provides a set of Integration Wizard functions that assist with the formatting and manipulation of complex types data.

The Process Orchestrator activators can use the ComplexTypesUtils support library which contains six functions. These are supported as follows:

Functions	Process Orchestrator for		
	.NET	Web Services	Java
FormatComplexType	✓	✓	×
FormatComplexTypeDirect	✓	✓	×
GetList	✓	✓	×
GetListDirect	✓	✓	×
GetNameValueList	✓	×	×
GetNameValueListDirect	✓	×	×

Table 17: Supported Complex Types Functions

Activated Java Beans do not have access to serialized objects by a .NET serializer (XML, SOAP or Binary) as Java supports its own serialization format. As a result none of these six functions apply. There is a workaround for GetList() which is documented below.

5.3.1 FormatComplexType

FormatComplexType returns a formatted string using a specified complex type value.

Parameters:

- **variableName** – specify a Metastorm Memo variable containing the serialized complex type value.
- **format** – manually define the return string format.

Returns:

FormatComplexType returns a formatted string.

FormatComplexType Example

In the example below, the function **GetClientRecord** is used to obtain the client record which is equal to the ID entered in the **intInputID** field.

memGetClientRecord stores XML. The XML is used by the **FormatComplexType** function to extract the client record details with user defined text into the **memFormatComplexType** field.

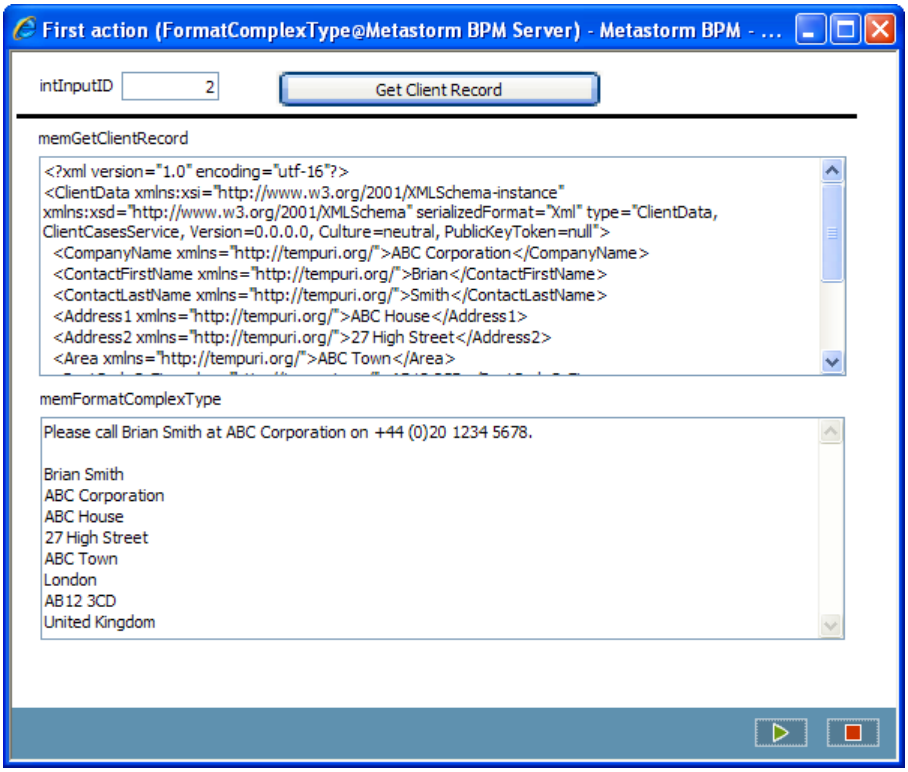


Figure 78: FormatComplexType Example

1. Create custom variables:

    a. Create a memo custom variable to store complex data (for example, memGetClientRecord).

    b. Create a memo custom variable for the formatted complex data (for example, memFormatComplexType).

2. Assign complex data to the memo custom variable, for example the return value from **GetClientRecord**.

Refer to section 5.1.3 Retrieving data into fields.

3. Create a memo field to display the return value.

4. Create an input field and a button to allow the user to input the Client ID.

5. Select the button and the **Do this** property tab.

6. Launch the Integration Wizard.

7. Select the **Action and form information** category and select **Assign value…**

8. Click **Next**.

 © Metastorm Inc., 2008

9. Select the **VariableName** field and select a memo custom variable, for example memFormatComplexType.

10. Launch the Integration Wizard from the **Value** field.

11. Select the **ComplexTypesSupportLibrary** category and select **FormatComplexType**.

12. Enter the **Variable** name which stores the complex data, for example memGetClientRecord

13. Enter the Formatted text in the **Format** field using < > for each simple data type.

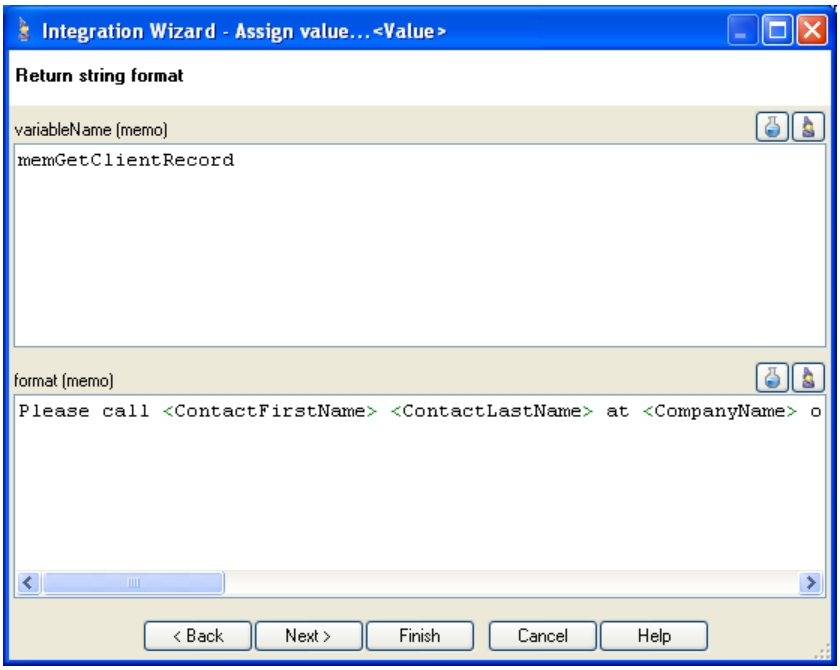


Figure 79: FormatComplexType Example

Text can be formatted, for example :

```
Please call <ContactFirstName> <ContactLastName> at <CompanyName> on
<Phone>. %LineFeed() %LineFeed()<ContactFirstName> <ContactLastName>
%LineFeed()<CompanyName> %LineFeed()<Address1> %LineFeed()<Address2>
%LineFeed()<Area> %LineFeed()<City> %LineFeed()<PostCodeOrZip>
%LineFeed()<Country>
```

14. Click **Finish**.

5.3.2 FormatComplexTypeDirect

FormatComplexTypeDirect returns a formatted string using a specified complex type value.

Parameters:

- **variableValue** – specify a value of a memo variable or a value which is obtained by calling another activated method and not stored in a memo variable field. The value must

contain a serialized complex type value. The serialized complex type value must be either an array or a collection object.

- **format** – manually define the return string format.

Returns:

FormatComplexType returns a formatted string.

FormatComplexType Example

In the example below, the function **GetClientRecord** is used to obtain the client record which is equal to the ID entered in the **intInputID** field.

memGetClientRecord stores XML. The XML is used by the **FormatComplexType** function to extract the client record details with user defined text into the **memFormatComplexType** field.

Using FormatComplexTypeDirect

The FormatComplexTypeDirect can be setup using two methods. The first method uses the steps explained above substituting in step 11 FormatComplexType with FormatComplexTypeDirect and in step 13 substituting memClientRecord with %memClientRecord.

The second method returns the memo data directly into the function and removes the need for two steps in the **Do this** property tab.

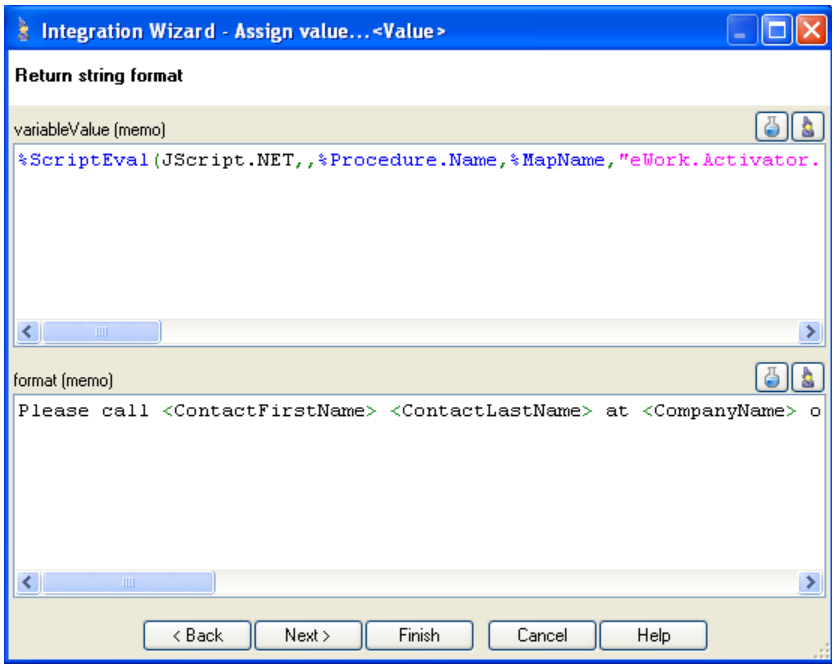


Figure 80: FormatComplexTypesDirect Example

5.3.3 GetList

GetList returns a delimited string using a specified complex type array or collection.

Parameters:

- **variableName** – specify a Metastorm Memo variable containing the serialized complex type value. The serialized complex type value must be either an array or a collection object.
- **item** – manually enter the name of public field or public property name. If a field name is not specified, the result of calling the `ToString()` method to an array or collection element will be used.
- **delimiter** – specify a string delimiter to separate the array or collection items. Default to a comma character.
- **start** – index position in the array or collection to start creating the delimited string. Default is 0.
- **end** – index position in the array or collection to stop creating the delimited string. Default is –1. When the start index is 0 all items in the array or collection will be selected. If the start index is not 0, then only a subset of items will be selected.

Returns:

GetList returns a delimited string.

GetList Example

In the example below **memReturnClients** stores the XML containing all client data. The **GetList** function is used to populate the **memListClients** field by setting **item** equal to company name.

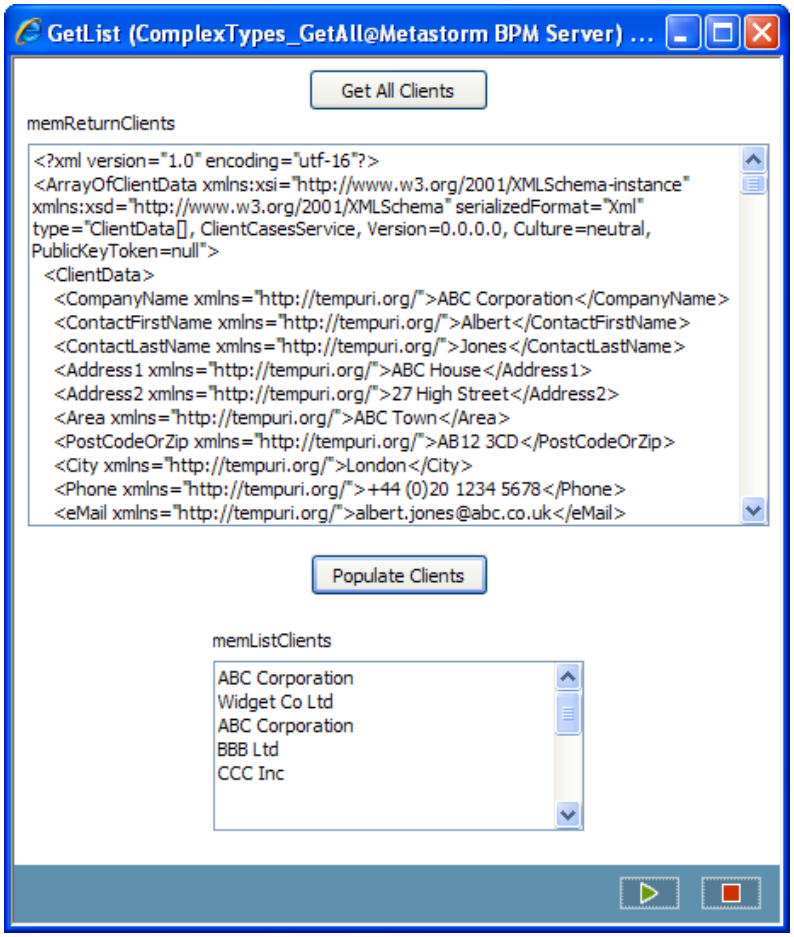


Figure 81: GetList Example

Java uses its own serialization format which is not compatible with XML, SOAP and Binary formats supported by .NET. Therefore, by default this function will not work. However there is a work around. A helper function EJB_GetListFromId() works in a similar manner to GetList() with the exception that its first parameter is a Java object ID instead of a .NET serialized object.

5.3.4 GetListDirect

GetListDirect returns a delimited string using a specified complex type array or collection.

Parameters:

- **variableValue** – specify a value of a memo variable or a value which is obtained by calling another activated method and not stored in a memo variable field. The value must contain a serialized complex type value. The serialized complex type value must be either an array or a collection object.
- **item** – manually enter the name of public field or public property name. If a field name is not specified, the result of calling the `ToString()` method to an array or collection element will be used.
- **delimiter** – specify a string delimiter to separate the array or collection items. Default to a comma character.

 © Metastorm Inc., 2008

- **start** – index position in the array or collection to start creating the delimited string. Default is 0.
- **end** – index position in the array or collection to stop creating the delimited string. Default is –1 which selects all items.

Returns:

GetListDirect returns a delimited string.

5.3.5 GetNameValueList

GetNameValueList returns a name and value delimited string using a specified complex type collection.

Parameters:

- **variableName** – specify a Metastorm Memo variable containing the serialized complex type collection.
- **item** – manually enter the name of public field or public property name. If a field name is not specified, the result of calling the `ToString()` method to an array or collection element will be used. The Metastorm Process Designer is required to specify the key and value type names of the collection items.
- **nameValueDelimiter** – specify a string delimiter to separate the name and value pair. Default is a tab character.
- **itemDelimiter** – specify a string delimiter to separate individual items. Default is a comma character.
- **start** – index position in the array or collection to start creating the delimited string. Default is 0.
- **end** – index position in the array or collection to stop creating the delimited string. Default is –1 which selects all items.

Returns:

A name and value delimited string.

This option is only available using an assembly activated using Process Orchestrator for .NET Activator.

5.3.6 GetNameValueList Direct

GetNameValueListDirect returns a name and value delimited string using a specified complex type collection.

Parameters:

- **variableValue** – specify a value of a memo variable or a value which is obtained by calling another activated method and not stored in a memo variable field. The value must contain a serialized complex type value. The serialized complex type value must be either an array or a collection object.

- **item** – manually enter the name of public field or public property name. If a field name is not specified, the result of calling the `ToString()` method to an array or collection element will be used. The Metastorm Process Designer is required to specify the key and value type names of the collection items.
- **nameValueDelimiter** – specify a string delimiter to separate the name and value pair. Default is a tab character.
- **itemDelimiter** – specify a string delimiter to separate individual items. Default is a comma character.
- **start** – index position in the array or collection to start creating the delimited string. Default is 0.
- **end** – index position in the array or collection to stop creating the delimited string. Default is –1 which selects all items.

Returns:

A name and value delimited string.

This option is only available using an assembly activated using Process Orchestrator for .NET Activator.

6 FORMS

The Designer allows you to create custom forms to meet your organization's requirements for collecting and distributing information. A form may include combinations of the following:

- A pre-defined form created through an external application.
- Formulas defining actions the system is to perform when the form is opened or closed.
- Background elements used to establish the layout and design of the form.
- Interactive foreground elements used to collect and display information.

Forms may be used to collect data, add it to the Metastorm database, and pass it to other users as part of a process. Form elements are the visual interface used to both gather and display this data.

Although any number of elements can be placed on a single form, too many cause clutter or crowding and become cumbersome for end-users. As a general rule, large forms with a lot of fields should be avoided, as they may be slow to publish and slow to display in the Designer and Clients. To avoid this, split the fields across a number of forms linked together by chained actions.

For further information on chained actions, refer to section 3.4.3Action-specific Properties.

Before adding elements to a form, it can be helpful to create a list or worksheet summarizing the information the form is to collect and display. This may help you to determine logical divisions of information that might be more usefully gathered or displayed on separate forms.

This section explains how to create a form, define its properties, and add background and foreground elements to it. In addition, instructions are provided for using external forms.

6.1 The Form Editor

Forms are created and modified in the Form Editor in the Designer Main Pane.

As you add elements to a form, you may find it necessary to manipulate the size and placement of them in relation to other elements. Metastorm Designer provides a basic set of tools to assist you in form layout and design.

6.1.1 The Form Toolbar

When the Form Editor is active, the Form toolbar is displayed:

Figure 82: The Form toolbar

The *Pointer arrow* button is used to select a particular form element.

The remaining buttons on the Form toolbar are used to add form elements to the form. The following table lists the buttons on the Form toolbar:

Button	Button Name	Button	Button Name
	Pointer arrow		Date/Time
	Image		Text
	Status		Drop-down
	Label		Radio group
	Rule/Frame		List
	Command Button		Memo
	Attachment Clip		Signature
	Check		Grid
	Number		Dataset
	Currency		Form Segment

*Table **18**: Form toolbar Buttons*

The form toolbar is dockable. It can be selected and dragged to the main pane where it forms a palette or it can be dragged to either side of the main pane where it docks in a vertical orientation.

6.1.2 Working with Form Elements

The **Edit** menu contains a number of options for working with form elements:

 © Metastorm Inc., 2008

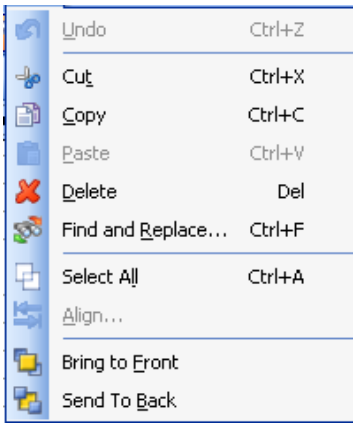


Figure 83: The Edit Menu

Depending on the form elements you have selected, different options may be unavailable in which case they are grayed out.

Adding Elements to a Form

You add a form element by clicking on the desired button on the Form toolbar and then clicking the location on the form where you want the element displayed. You may also add an element and then change the type through the Properties editor, if necessary.

To add several copies of a particular form element, hold down the Shift key while clicking on the desired element button, then click at each location on the form where you want the elements placed. Click on the Pointer arrow or select another form element to disable the multiple placement function.

Selecting Form Elements

To move or resize a form element, or to display the corresponding Properties editor, you must select it.

You can select individual form elements by clicking on the Pointer arrow button on the Form toolbar and then clicking on the form element. The selected element will have handles at each corner and on all sides, which may be used for moving or resizing it.

Multiple elements may be selected using any of the following methods:

- From the **Form** toolbar, choose the **Pointer arrow** button. Place your cursor over a point on the form, click on the left mouse button to change the cursor to a crosshair, and, holding the left mouse button down, drag your cursor to encompass all of the desired form elements. Designer will display a dashed line indicating the outer edges of your selection. All items within the dashed line will be selected when you release the mouse button.
- From the **Form** toolbar, choose the **Pointer arrow** button, and click on a form element. Then, holding either the **Shift** or the **Ctrl** keys, click on any additional elements desired to select them.
- From the **Edit** menu choose **Select All**. Designer will select all elements currently on the form.

If you have selected multiple form elements, the handles surrounding each selected element will be grayed-out. You may move the elements as a group to another position on the form as well as cut, copy, or delete the group. However, you may not resize the elements as a group.

Moving Form Elements

Form elements may be moved by selecting them and dragging them to a new location.

Copying and Pasting Form Elements

Form elements may be copied by selecting the desired element and then selecting the **Copy** button from the General toolbar, or by using the **Edit | Copy** option.

Form elements may be pasted onto a form by selecting the **Paste** button from the General toolbar, or by using the **Edit | Paste** option.

Aligning Form Elements

Individual and groups of form elements may be centered horizontally or vertically on the form. Groups of form elements may be aligned vertically or horizontally relative to each other through the **Edit | Align** option. The following table shows the selections available through the Edit/Align option:

Horizontal	*Vertical*
No change	No change
Left sides	Tops
Centers	Centers
Right sides	Bottoms
Space equally	Space equally
Center in window	Center in window

*Table **19**: Form Element Alignment Options*

Metastorm Designer does not consider captions when it aligns form elements; alignment is based upon the element's data field size and position. You should inspect your form layout to verify that the captions for each form element appear as desired.

Determining Form Element Layering

Form elements are divided into two groups: foreground and background elements. Foreground elements *always* overlay background elements. However, individual elements within each group may be brought to the front or sent behind another element using the **Bring To Front** and **Send To Back** options from the Edit menu.

To place one form element in front of another, select the form element, and from the **Edit** menu select **Bring To Front**.

To place one form element behind another, select the form element you want to send to the back, and from the **Edit** menu select **Send To Back**.

 © Metastorm Inc., 2008

Metastorm Designer does not consider captions when it layers form elements; layering is based upon the element's data field position. You should inspect your form layout to verify that the captions for each form element appear as desired.

Windowed controls (such as frames) always stack on top of non-windowed controls (labels are the only non-windowed controls used in the Designer), so you can change a label's stacking order only in relation to other labels.

Deleting Elements from a Form

Form elements may be deleted from a form using any of the following methods. Select the desired element and:

- Press the **Delete** key on your keyboard.
- Use the **Edit | Delete** menu option.
- Use the **Edit | Cut** option.
- Click on the **Cut** button.

Form elements removed using the Cut methods are saved in the paste buffer, and may be pasted back onto the form, or onto another form in the same procedure. Form elements deleted using the Delete options (Edit | Delete, or the Delete key) are not stored. However, you may use the Edit | Undo option to restore deleted items immediately.

Placing Form Segments on a Form

When placing a form segment on a form, the elements (fields etc.) will appear on the form offset from the top left hand corner of where the form segment is placed, by the same offset from the top left hand corner of where they are on the form segment.

For example, if a date field is located at position (10,10) on a form segment, and that form segment is positioned on a form so that it's top left corner is at position (15,20) on the form, the date field from the form segment will display at position (25,30) on the form.

Form segments cannot be placed in other form segments.

Bear in mind that the more form segments there are on a form, the longer the form will take to display in the Designer and to publish. It is recommended that a single form segment contains no more than 10 to 12 fields, and that a single form includes no more than 3 or 4 of these segments

6.2 Properties editor

Each of the elements of a form has certain default properties; however, in order for your process to function properly, you may find it necessary to modify some or all of these properties. This is done through the Properties editor.

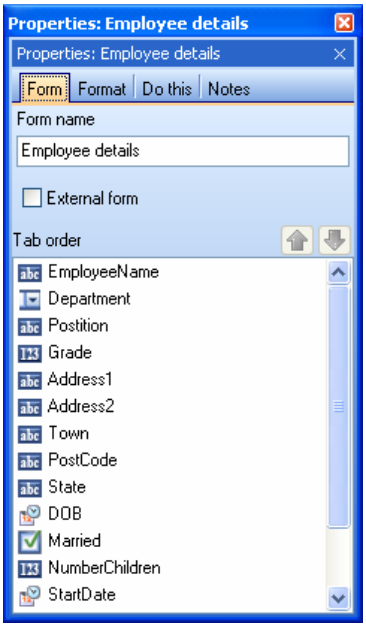


Figure 84: Properties editor

Each form element has a Properties editor tab specific to the type of element. Other tabs of the Properties editor are common to several different form elements, but may display different property fields, or require instructions based on the type of form element you are working with. This section is organized by form elements, and includes instructions for working with all Properties editor tabs for each type of form element individually.

The Properties editor can be set to a floating dialog, or it can be docked to the sides or top or bottom of the main pane. When docked to the main pane perimeter a push-pin is available that sets the editor to hide when not selected. The Properties editor can be toggled on or off using:

- The buttons on the General toolbar;
- The **View** menu: or
- Right click on the form and select **Properties** from the popup menu that appears.

Many of the properties you can modify through the Properties editor require you to enter a formula. In most cases these fields will include options for calling on the Integration wizard and the Formula editor.

Instructions for using these tools may be found in Section 10, Creating Formulas, page 261.

6.3 Creating a New Form

In most procedures, you will find it necessary to create one or more forms. These forms will be used to collect or distribute information.

Forms may consist of the following elements:

- A form background, provided by default (required)
- Background elements, such as images, frames, and rules, which are used as layout and design elements (optional)

 © Metastorm Inc., 2008

- Foreground elements, such as command buttons, attachment clips, grids, and data fields, which are used to collect and display information.

In addition to any form elements you choose to add, forms have several predefined properties, which may be changed or reviewed as required using the Properties editor.

Form names can use characters which are configured with a default locale of North American, Western European or Central European character sets. However, form names cannot include any of the following characters: Pipe (|), Period (.) Dollar Sign ($), Equal Sign (=), Back Slash (\), Quotation Mark (“), Forward Slash (/), Percent (%), Comma (,), Plus (+), Colon (:), Semi-colon (;) and Angle brackets (<>).

6.3.1 Defining Form Properties

To create a new form (or Administration form):

- Click on the **New Form** (or **New Administration Forms**) button in the General toolbar; or
- From the **Component** menu select the **New Form** (or **New Administration Forms**) option.

Designer places an empty form in the Form Editor and displays the Form toolbar. A form item is added to the Procedure Explorer.

The Properties editor displays the following tabs for defining form properties:

- Form.
- Format.
- Do this.
- Notes.

Defining a Form’s Form *Properties*

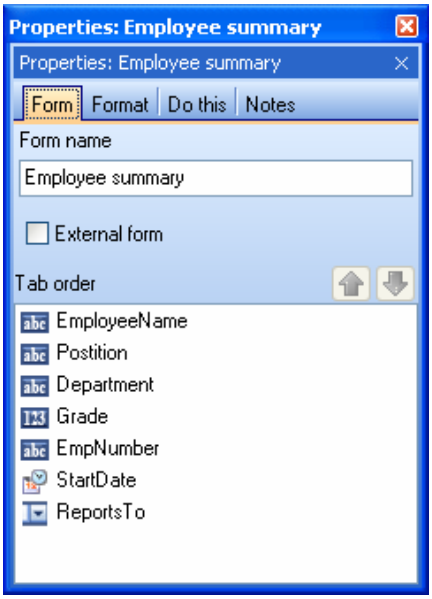


Figure 85: Form Tab

General properties for a form are defined on the **Form** tab of the Properties editor.

On the **Form** tab of the Properties editor, enter a short, meaningful name for the form in the **Form name** field. The following characters are not supported in the *Form Name* field:

- Pipe (|).
- Period (.).
- Dollar Sign ($).
- Equal Sign (=).
- Back Slash (\).
- Quotation Mark (“).
- Forward Slash (/).
- Percent (%).
- Comma (,).
- Plus (+).
- Colon (:).
- Semi-colon (;).
- Open bracket (<).
- Close bracket (>).

The name is displayed when the user opens a folder that includes this form. The name field may not be left blank. The form name must be unique for each form in a single procedure.

A number of words have special meanings in the databases that Metastorm BPM supports. You should not use any of these reserved words when naming a form. See the Reserved Words.txt file for a list of reserved words. This file is located in the Metastorm BPM\Designer folder.

To use a form, leave the **External form** check box *unchecked*.

For information about using an external form in place of a Metastorm form, refer to page 224.

The **Tab order** field shows a list of all foreground elements in the form (button, clip, data field or grid). This list will be updated as new elements are added to the form. By default these will be listed in the order in which they are created.

To specify the order in which foreground fields receive focus when the Tab key is pressed at run-time, drag and drop the items until they are in the order you require. The Tab key will ignore fields defined as read-only.

 © Metastorm Inc., 2008

Defining a Form's Format *Properties*

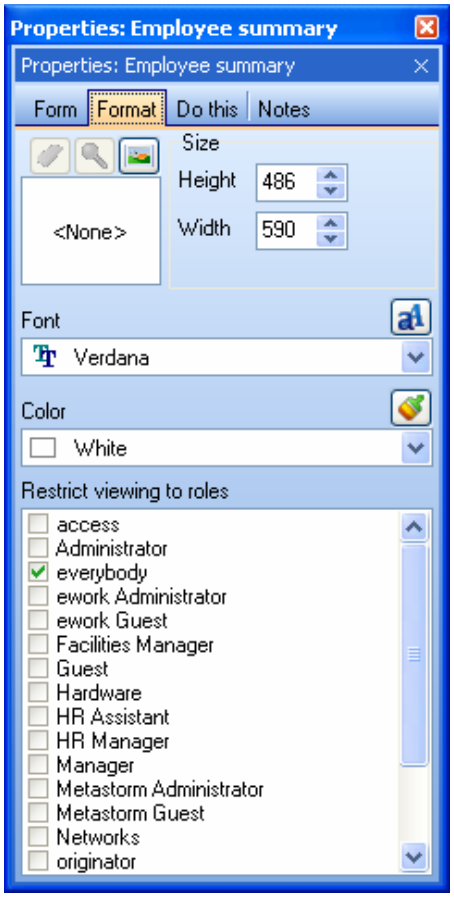


Figure 86: Format Tab

The base appearance of a form (background images, color, and fonts) is determined through the **Format** tab of the Properties editor. The remainder of the form's appearance is determined through the layout (placement) of form elements.

19. To define a background image for the form, click on the **More Pictures** button above the image viewing area. Designer will display a standard **File Open** dialog, allowing you to browse for the image you wish to use.

    When the image is selected it will display in the image viewing area and the **Clear** and **Preview** buttons will become available. If no image is selected the viewing area shows "(None)."

    To see a preview of the selected picture as a background, click the **Preview** button.

    To delete a background image from the form, click on the **Clear** button.

20. The Form **Size** property (in pixels) includes two fields: **Height** and **Width**.
    The default height of the form is 320 pixels. To change this initial height, make your selection from the **Height** field.

    The default width for the form is 512 pixels. To change this initial width, select a value from the **Width** field. The form can also be resized by dragging its borders.

The user may also resize the form when they view it, although placement of form elements will remain consistent with your design.

21. To select the default font for the form, scroll through the available drop. Select the font format button to change font color, style and size.

If you anticipate that your users may be viewing your form through the Windows clients, we recommend that you design your form layout using the "large fonts" setting on your Windows system.

If you anticipate that your users may be viewing your form through a Web browser, keep in mind that the underline and strikeout options are not supported in Web browsers.

22. To select a background color for the form, click on the **More Colors** button . The Designer will display a color palette from which you may select a background color.
23. In the **Restrict viewing to roles** field, select those roles that should be allowed access to the form. All of the roles available for this procedure are included in this list. In addition, the following default roles are included in this list and have special functions:
    - *Access* – represents the role chosen for the map's *Limit access to roles* property. Only users with the chosen role will have access to this form.
    - *Everybody* – grants access to this form to any user registered on the Metastorm system.

For further information on using the 'everybody' role, refer to section 8.1.8, Using the 'everybody' role

    - *Metastorm Administrator* – grants access to this form to individuals holding the "Administrator" role on the Metastorm BPM system.
    - *Metastorm Guest* – grants access to this form to anyone registered as "guests" on the Metastorm system.
    - *Originator* – grants access to this form to the individual who originated the folder.
    - *To Do list* – grants access to this form to any users who have the folder in their To Do list.
    - *Watch list* – grants access to this form to any users who have the folder in their Watch list.

If a user does not hold any of the selected roles, the form will not appear in a folder. Also, a form with no roles selected on its ***Restrict viewing to roles*** *property is available to everybody.*

Existing Procedures that have been migrated from Version 6 to Version 7 may use e-Work Administrator and e-Work Guest as the equivalent of Metastorm Administrator and Metastorm Guest.

 © Metastorm Inc., 2008

Defining a Form's Do This *Properties*

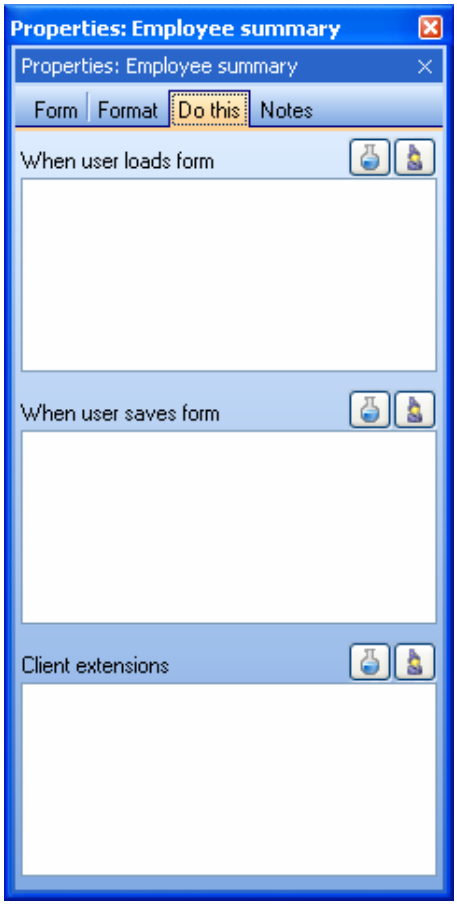


Figure 87: Form's Do this Tab

Actions that should be performed when the form is opened or closed are defined on the **Do this** tab of the Properties editor.

1. In the **When user loads form** field, enter any commands the system should perform when the form is opened, whether as a tab in a folder (to be reviewed) or as an action form (to be initiated or modified). You may create the or edit the formula using either of the following tools:
    - Click on the **Integration wizard** button to open the Integration wizard.
    - Click on the **Formula editor** button to open the Formula editor dialog.

*If the form is being used as part of an action, the **When user loads form** command will be performed after the action's* When action started *command.*

*You should not use the **When user loads form** property to set field defaults for a blank form. This should be done using the* When action started *property.*

2. In the **When user saves form** field, enter any commands the system should perform when a user commits (saves) an action that uses this form, whether as a page in a folder or as an

action form. You may create the formula (and edit it later, as necessary) using either of the following tools:

- Click on the **Integration wizard** button to open the Integration wizard.
- Click on the **Formula editor** button to open the Formula editor dialog.

If the form is being displayed as the result of a user taking an action, commands will be performed in the following order: (1) When action started *(2)* When user loads form, *(3)* When user saves form, *(4)* When action completed. *If the form commands in the* When user saves form *property fail, the form will not be closed and the action will not be committed. The user may choose to change the data and try again or cancel the action.*

3. In the **Client extensions** field, enter any commands the system must perform when the form is loaded (whether as a page in a folder or as an action form), submitted or cancelled. You may create the formula (and edit it later, as necessary) using either of the following tools:

   - Click on the Integration wizard button to open the Integration wizard.
   - Click on the Formula editor button to open the Formula editor dialog.

Defining a Form's Notes *Properties*

This tab of the Properties editor displays a memo field that the process designer may use to make designer notes. These notes will not be seen by end-users.

6.4 Form Element Properties

Form elements are the background and interactive items you can place on a form. These elements are selected using the Form toolbar.

For details refer to Table 18, in Section 6.1.1.

When Designer places a form element on the form it displays the corresponding Properties editor.

6.4.1 Dependencies

Dependency properties for form elements are set on the *Do This* tab of the form element's Properties editor. The following options are available to any fields mapped to a folder or custom variable (that is, any data field that can be used for input when the form is used in an action):

- Field is dependent on another

  Checking this box tells the Process Engine that, whenever a new value is received from a client, the Process Engine must recalculate the value of this particular field.

- Field has dependants of its own

  Checking this box tells the Metastorm client (or browser) that, whenever the value in the field is changed by the user, the new value needs to be sent immediately back to the Process Engine so that the form can be refilled with new calculated values. (With any other input field, the value is sent back only when the user submits an action form.)

These fields appear on the *Do this* tabs for the following data fields' Properties editors:

 © Metastorm Inc., 2008

Check	Text	List	Currency
Number	Dropdown	Memo	
Date/Time	Radio group	Signature	

Unmapped data fields, status fields, grids, or datasets have only the **Field is dependent on another** *check box.*

6.4.2 Client Extensions

Client Extensions are used to specify client-specific behavior. The **Client Extensions** field of the Properties editor *Do this* tab allows a process designer to specify client side script functions called when a field is entered or left; when a form is loaded, submitted or cancelled; when a button is pressed; or when a grid cell is left. You may create or edit the formula using either of the following tools:

- Integration wizard, opened by clicking on the Integration wizard button.
- Formula editor dialog, opened by clicking on the Formula editor button.

White spaces should not be included in formulas entered into the Client Extensions field.

6.4.3 Defining Element-independent Properties

The following form properties tabs are element-independent and are common to all form elements:

- Extra.
- Notes.

Defining a Form Element's Extra *Properties*

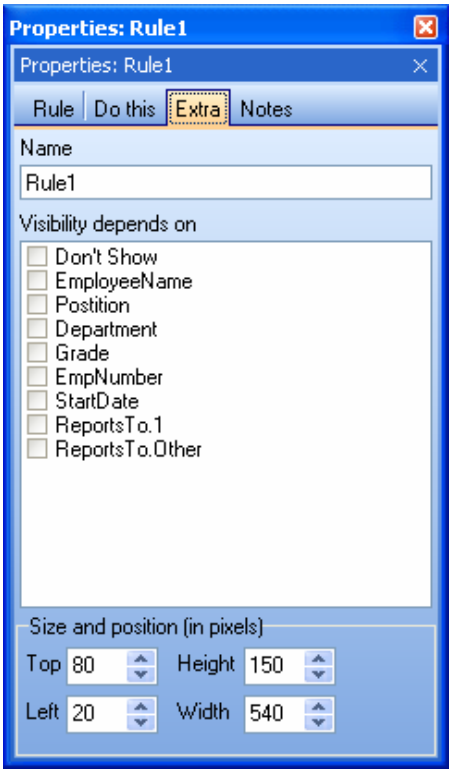


Figure 88: Extra Tab

The **Extra** tab allows you to define the position and usually the size of a form element, as well as specify whether it should be displayed at all times or only under certain conditions.

1. Each form element is created with a default name based on its type (for example, "*List1*"). You can change this if you wish by entering a new name in the **Name** field. This field may not be left blank, and may not include non-US English characters.

   A number of words have special meanings in the databases. You should not use any of these reserved words when naming a form element. See the Reserved Words.txt file for a list of reserved words. This file is located in the Metastorm BPM\Designer folder.

   When designing a form that will be exposed as a web service, take care not to use VBScript reserved keywords in the fields' names.

2. If you want the form element to be displayed only under certain conditions, select the appropriate options from the **Visibility depends on** field.

   Dataset and Form Segment fields do not have a ***Visibility depends on*** *field.*

   This property displays a list showing the name of each field on the form. The conditions for each type of field are as follows:

   - **Don't show**. The current form element will not be displayed on the form.

 © Metastorm Inc., 2008

- A **check box**. The current form element will be displayed only if the check box identified here has been checked.
- A **currency** field**.** The current form element will be displayed only if the currency field selected here contains a non-zero value.
- A **number** field. The current form element will be displayed only if the number field selected here contains a non-zero value.
- A **radio group**. There is no entry for the radio group as a whole; each option is listed individually by field name and line number (from 1 upward). The current form element will be displayed only if the radio group item identified here has been selected.
- A **drop-down** field. There are two entries for each drop-down field. The first (comprised of the field name, followed by a period and the number 1) indicates that the form element is to become visible if the *first* entry in the field's drop-down list is selected. The second entry (comprised of the field name, followed by a period and the word *other*) indicates that the foreground form element is to become visible if any item *other than the first* in the field's drop-down list is selected.
- A **date/time**, **list**, **text**, or **memo** field. The form element will be displayed only if the field contains information. For example, the form element will be visible only if a date/time has been selected, an item selected from a list, or data entered into a memo field.
- A **status** field. There are three entries for each status field. They comprise of the field name followed by Safety, Warning or Danger. If, for example, `Status1.Safety` is checked, the current form element will be visible if the status field, Status1, is in the Safety state. The three states are not mutually exclusive. If the visibility of a field is set to depend on all three states then the field will always be visible.

If one or more entries are checked, then the form element will be shown only when each checked field contains a value.

3. The **Size and position** (in pixels) property contains the following options:

   - **Top**. Use this field to position the top of the form element within the form in pixels.
   - **Left**. Use this field to position the left edge of the form element within the form in pixels.
   - **Width**. Use this field to select the width of the form element in pixels.

   When this option is used to add width to a vertical rule, the rule is converted to a frame.

   This option crops an image, check box or radio group caption area from the right side or adds blank space to the right side in excess of the field's actual size. It does not resize the image, check box or radio group.

   - **Height**. Use this field to specify the height of the form element in pixels.

   When this option is used to add height to a horizontal rule, the rule is converted to a frame.

   This option crops an image from the bottom or adds blank space at the bottom of the image in excess of the image's actual size. It does not *resize an image.*

You can also adjust the position of a form element by clicking and dragging it to the desired location on the form. You can resize the background form element (or crop it from the right or bottom, if it is an image) by clicking and dragging on the handles surrounding the background form element until it reaches the desired size (or is cropped as required).

You can *Resize* a command button, date/time, drop-down, list, memo, number, radio group, currency, signature, text or grid field (from the right side or from the bottom) by clicking and dragging on the handles surrounding the form element until it reaches the desired size.

Note: you cannot adjust the height of date/time, drop-down, number, currency and text fields.

Adjusting the width (and height, for a label) of a label, drop-down, list, memo, number, radio group, signature or text field does not take into account the field's caption area or list items. You should check your form to ensure captions are positioned as required.

A command button's caption is automatically centered on the button. As you resize the button, check your form to verify that the caption displays properly.

You cannot adjust the height or width of a status field.

If a grid width is too small to display all columns, Designer creates a horizontal scroll bar at the bottom of the field. You may not make a grid so small that the column name row is not visible.

If a list field is too short to display the entire list, a vertical scrolling bar is created.

Note that:

- An attachment clip or a dataset field does not have a **Width** or **Height** field.
- You can neither resize nor crop an attachment clip or dataset field.

Defining a Form Element's Notes *Properties*

The Notes of the Properties editor displays a memo field that the designer may use to make design notes. These notes will not be seen by end-users.

6.4.4 Defining Element-specific Properties

All form elements have element-specific properties as well as the element-independent properties discussed in section 6.4.3. This section will address each form element's specific property tabs. A general description of the form element is provided followed by a detailed presentation of each of the element's specific property tabs.

Defining a Frame or a Rule

You can place any number of frames and rules on a form to divide it into separate areas.

Click on the **Rule/Frame** button on the form toolbar, and then click on the location on the form where you want the frame to be displayed. Designer places a frame on the form, and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining Frame/Rule properties:

 © Metastorm Inc., 2008

- Rule
- Do This
- Extra
- Notes

For details of the Extra tab and the Notes tab, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Frame/Rule's Rule *Properties*

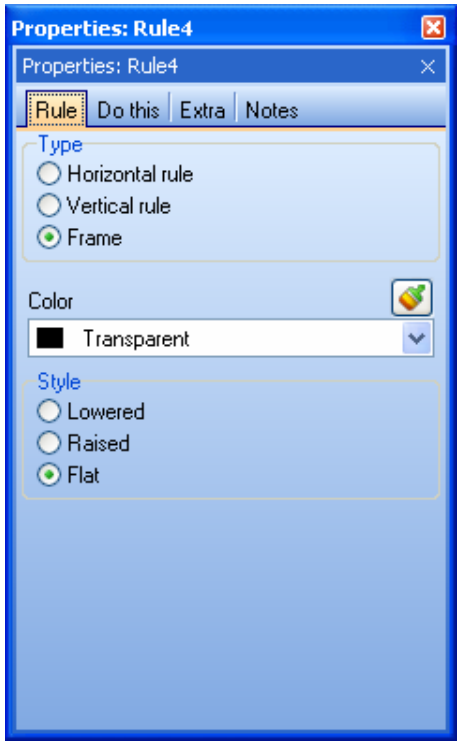


Figure 89: Rule Tab

The Rule tab contains a **Type** radio group allowing you to select from the following options:

- Horizontal rule
- Vertical rule
- Frame

Metastorm BPM displays the type of rule you select.

Use the drop-down list in the **Color** field to change the default color value, or click the **More Colors** button for additional color options.

Set the style of the frame or rule by selecting the desired option from the **Style** radio group.

Defining an Image

You can place any number of images on the background of a form.

Click on the **Image** button on the form toolbar, and then click on the location on the form where you want the image to be displayed.

Metastorm Designer will display a blank image field and the corresponding Properties editor.

The Properties editor displays four tabs for defining Image properties:

- Image
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining an Image's Image *Properties*

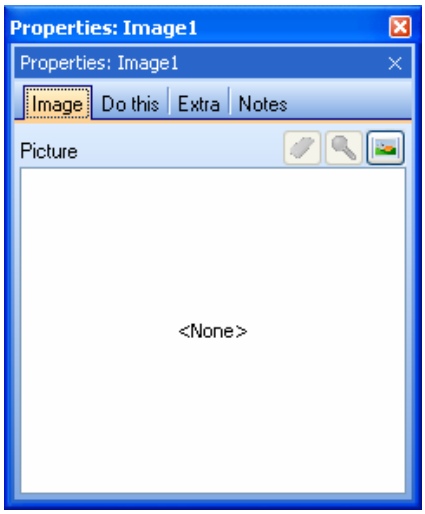


Figure 90: Image Tab

1. The Image tab contains an empty **Picture** field where a copy of the image you select will be displayed. To specify the image to be added to the form, click on the **More Pictures** button or double-click in the picture viewing area.
    - Designer displays the standard File Open dialog box.
2. Browse for the file you want. When an image is selected, it will display in the Picture field.
3. To preview the picture on the form, click the **Preview** button above the viewing area.
4. To remove an image from the form, click on the **Clear** button above the viewing area. The image will be erased from the Properties editor and will also be removed from the form when the form is returned to focus.

When you publish the procedure, Designer stores the image file in the database as an application attachment, places a copy of the image in the image field of the Properties editor, and updates the form to display the image.

 © Metastorm Inc., 2008

Defining an Image's Do This *Properties*

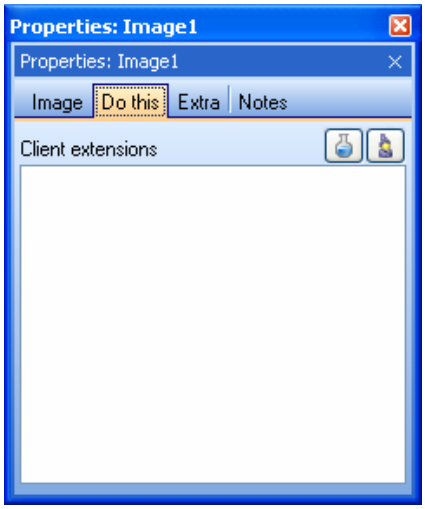


Figure 91: Image / Do This Tab

An image has only one item on the *Do This* tab of the Properties editor.

In the **Client extensions** field enter any commands the system must perform when the form is entered or exited, whether as a tab in a folder or as an action form. You may create and edit the formula using either of the following tools:

- Integration wizard, opened by clicking on the Integration wizard button
- Formula editor dialog, opened by clicking on the Formula editor button.

Defining a Label

You can place any number of labels on the background of a form to display titles or fixed captions.

To create a label, click on the **Label** button on the form toolbar, and then click on the location on the form where you want the label to be displayed.

Designer will display a label field and the corresponding Properties editor.

The Properties editor displays four tabs for defining Label properties:

- Label
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Label's Label *Properties*

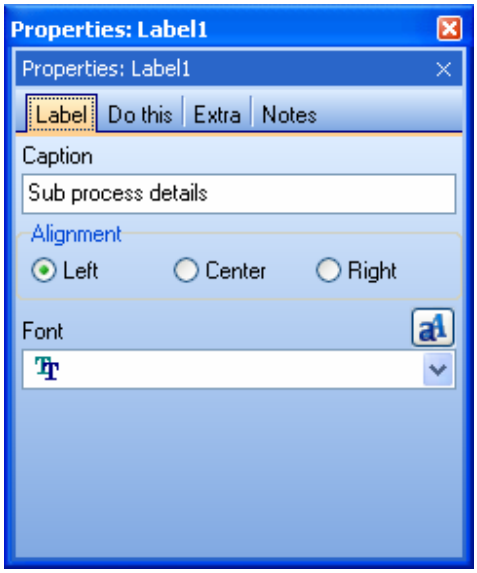


Figure 92: Label Tab

1. To select a different font, use the drop-down arrow to display the available fonts. To change the font style click the **More Fonts** button to view additional options.

Designer updates the form to display the caption in the font selected for the form.

2. The **Alignment** radio group allows you to select from the following options:

   - **Left** – The caption text will be left-justified within the caption field.
   - **Right** – The caption text will be right-justified within the caption field.
   - **Center** – The caption text will be centered within the caption field.

   The caption text will wrap within the caption field. If the caption field is too small, the portions of the text that extend beyond the field boundaries will not be visible. You should check your form to verify that the label field is large enough to display all of the text, and resize it as necessary.

3. A sequentially numbered default caption for the field type is displayed when each new label field is added to the form. To change the default caption, type the label's caption text that to be displayed on the form.

Designer updates the form to display the new caption text.

 © Metastorm Inc., 2008

Defining a Label's Do This *Properties*

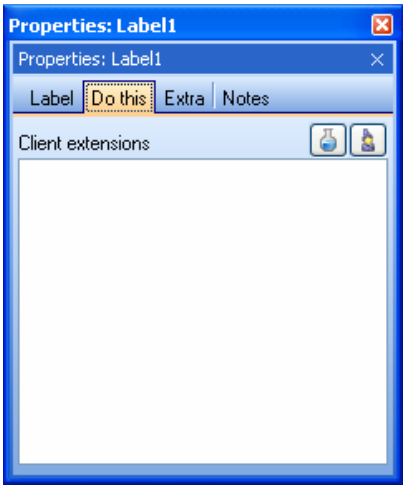


Figure 93: Label / Do This Tab

A label has only one item on the *Do This* tab of the Properties editor.

In the **Client extensions** field, enter any commands the system must perform when the form is entered or exited, whether as a page in a folder or as an action form. You may create or edit the formula using either of the following tools:

- Click on the Integration wizard button to open the Integration wizard
- Click on the Formula editor button to open the Formula editor dialog

Using a Label field to add a link to the form

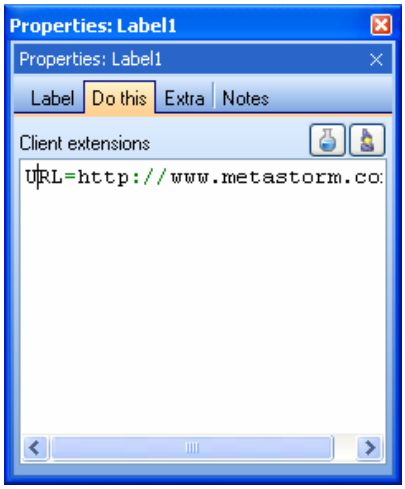


Figure 94: URL set in Do this tab

You can add a URL link to a form using a label field. To achieve this:

1. Add a label field to the form.
2. Set the **Caption** field on the *Text* tab. The link that appears on the form will have this caption.
3. On the *Do This* tab, set the **Client Extensions** to 'URL=<your chosen URL>'. For example, 'URL=http://www.Metastorm.com'.

This works only for the Web Client, or Windows Clients using web forms.

Using a Label field to add HTML elements to the form

You can embed small fragments of HTML into a form using the caption of a label field. To do this:

1. Add a label field to the form.
2. Enter HTML commands in the **Caption** field on the *Text* tab. For example:
   - `<a href="http://www.metastorm.com" target="_blank">Go to Metastorm's web site</a>`

     This HTML adds a link to the Metastorm website to the form. The caption will read 'Go to Metastorm's web site'.
   - `<iframe src="http://www.metastorm.com"></iframe>`

     This HTML embeds a view of the Metastorm web site within the form.

Any HTML embedded in a form in this way becomes part of the larger HTML document that is used to generate the form. Therefore there are limits to the HTML that can be embedded using the Label field.

For example, the HTML that makes up the form contains a <BASE> element. This contains a base HREF. An <A> element cannot be used with this <BASE> element unless it references an external file, as in the following example: `<a href="http://www.metastorm.com"/>`. Therefore some <A> elements will not work in HTML embedded in a Metastorm form.

When embedding HTML in a form, be aware that the embedded HTML may behave unpredictably when run as part of the HTML for the form.

 © Metastorm Inc., 2008

The figure below shows a form containing an embedded view of the Metastorm web site, and two links to the same web site. The first link is achieved by embedding HTML, while the second makes use of the client extensions.

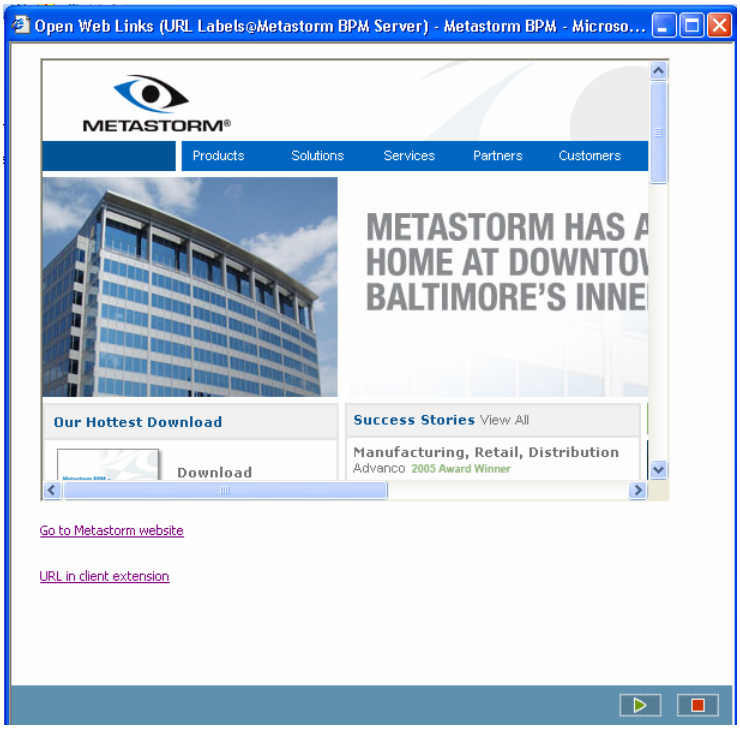


Figure 95: Form containing embedded HTML and links

It is not possible to change the embedded HTML at runtime.

Defining an Attachment Clip

An attachment clip is an interactive form element used to attach a file to a form. You can place any number of attachment clips on the foreground of a form to hold attachment files for each folder. However, if you are anticipating more than one attachment to be made to the form, set the clip to **Multiple Attachments**.

There is no advantage to adding a single attachment clip and a multiple attachment clip to the same form, as multiple attachment clips contain all attachments for a folder.

*For further information about the option **Document Management Support**, refer to section 7 Document Management Support.*

To add an attachment clip, click on the **Attachment clip** button on the form toolbar and then click on the location on the form where you want the Attachment clip icon to be displayed.

Designer places an attachment clip on the form and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining Attachment clip properties:

- Clip

- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

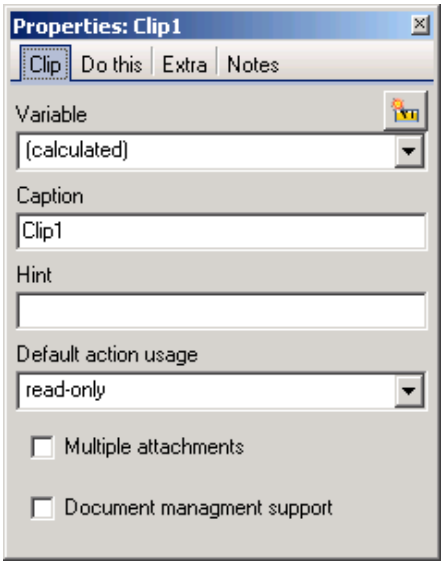


Figure 96: Clip Tab

1. Click on the **Multiple attachments** check box to allow multiple attachments to be added to the form. The clip on the form changes to a grid on which the user may add one or more attachments. Only one multiple attachment grid should be placed on a form. If you choose the multiple attachments option, not all the properties are available in the Properties editor. Where this is the case, it is noted below.
2. The Designer will use the value in the Variable property for the caption by default. You may enter a new caption in the **Caption** field, if desired.

   A multiple attachment grid does not have a caption.

3. Select the name of the variable associated with this attachment clip in the **Variable** field. There are a number of options:

   - If the attachment name is to be calculated at run-time, select *calculated* from the list. The calculation returns the name of a clip, which must already exist – because the field is read-only, you can't save a new attachment to the clip. You will be able to specify how it is calculated on the *Do this* tab.
   - If the attachment name is to be stored in the database, you can select a variable from the drop-down list. Only text variables will appear in the list.
   - You can create a new custom text variable by clicking on the **New custom variable** button and specifying the details of the variable.

 © Metastorm Inc., 2008

The text variable must be large enough to hold the file name of the attachment. It is strongly recommended that the text variable be defined with a length of 250 characters as the process designer has no control over the file names that a user will place on a clip.

The variable you select on this tab of the Properties editor will determine the fields that appear on the attachment clip's *Do this* tab. Designer stores the variable in the database table for the map the associated with the current form, and stores the attachment itself in the attachment table.

You cannot associate a variable with a multiple attachment grid.

4. You can include a text string to give users a brief help message when the cursor rests on the attachment clip icon. Type the message you want displayed in the **Hint** field. This hint will be displayed when the clip field has no file on it. Once a file has been attached to the clip field, the hint will display the name of the file.

5. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Attachment Clip.

Defining an Attachment Clip's Do This *Properties*

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the attachment clip change.

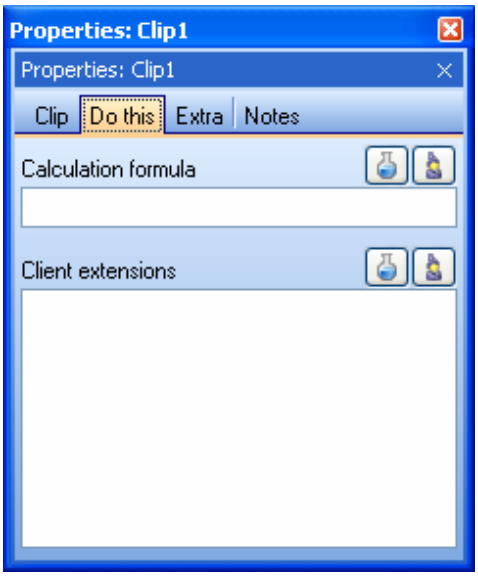


Figure 97: Clip / Do This Tab

1. The **Calculation formula** field is displayed only if you have selected a *calculated variable*. Specify the variable or formula to be used to obtain the attachment name in this field. You may create and edit the formula using either of the following tools:
   - Click on the Integration wizard button to open the Integration wizard.
   - Click on the Formula editor button to open the Formula editor dialog.

In the case of an attachment clip, the value is only the attachment name, and not the attachment itself.

You cannot associate a Calculation formula with a multiple attachment grid.

If a custom variable has been assigned to the attachment clip, the *Do this* tab will display a **When Changed** option instead of a calculation formula.

In the **When changed** field, *enter* any command the system is to perform when the contents of this clip field are changed.

*The clip field's **When changed** event can fire only when the form is submitted. **When changed** events will fire only when the form is submitted if the Engine is set up to Fire When Changed On Submit. For information, refer to the Administration Guide.*

A multiple attachment grid does not have a When Changed option.

2. Enter any commands to be performed by the system when this clip field is entered or exited in the **Client extensions** field. You may create and edit the formula using either of the following tools:

    - Click on the Integration wizard button to open the Integration wizard
    - Click on the Formula editor button to open the Formula editor dialog.

Client Extensions on attachment clips and multiple attachment grids are not supported when using Web forms.

Defining a Command Button

You can place any number of command buttons on the foreground of a form, so that the user can invoke actions on the client or the server.

To add a command button, click on the **Command button** icon on the Form toolbar and then click on the location on the form where you want the button displayed.

Designer places a Command button on the form and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining Command button properties:

- Button
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

 © Metastorm Inc., 2008

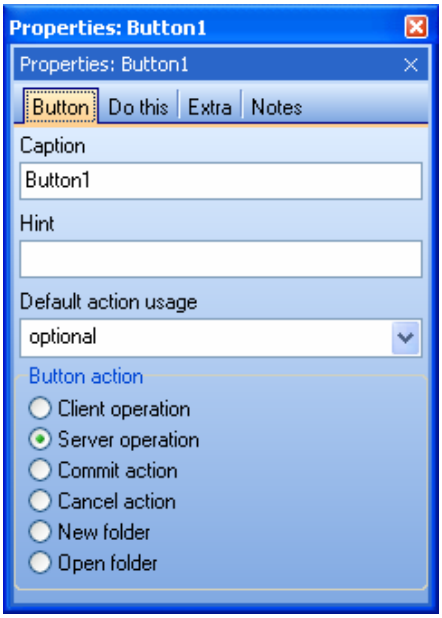


Figure 98: Button Tab

Defining a Command Button's Button *Properties*

1. The **Button action** field contains a radio group listing the different types of button actions available:
    - **Client operation** – Uses the *Client extension* formula (which you define on the *Do this* tab) to perform an operation specified by the formula.
    - **Server operation** – Uses the *When button pressed* formula (which you define on the *Do this* tab) to perform an operation specified by the formula.
    - **Commit action** – Updates the *database* with any changes made to the form and may move the folder to the next stage.
    - **Cancel action** – Allows you to *cancel* the form without saving any changes.
    - **New folder** – Allows you to initiate a new blank form associated with this or another map.
    - **Open folder** – Allows you to open a folder created previously by this or another procedure, as specified in the *Open folder* field (on the *Do this* tab).

Select the type of button action desired.

2. You can include a text string to give users a brief help message when the cursor rests on the button. Type the message you want displayed in the **Hint** field.

3. Enter the caption to be displayed on the button in the **Caption** field. If you choose not to name the button, it will be displayed with a default caption of "Button" plus an auto-incremented number.

4. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Command button. Use the down-arrow associated with this field to select the desired default.

Defining a Command Button's Do This *Properties*

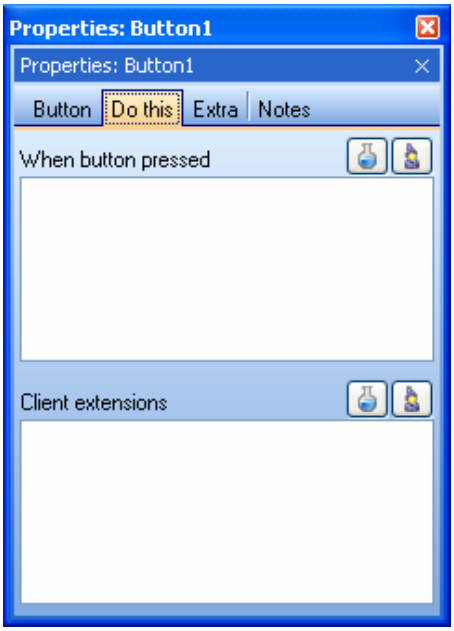


Figure 99: Command Button / Do this Tab

The items displayed on the *Do this* tab of the Properties editor vary depending upon the type of button action selected on the *Button* tab. The following table lists the possible Do This tab properties:

Button Action	**Do this *Tab Properties***
Client operation	Client extensions
Server operation	When button pressed Client extensions
Commit action	Client extensions
Cancel action	Client extensions
New folder	New folder action New folder map Client extensions
Open folder	Open folder Client extensions

Table 20: Possible Do This tab properties

- **When button pressed**

  In this field, enter any commands the system must perform when the button is pressed. You may create and edit the formula using either of the following tools:

  - Click on the Integration wizard button to open the Integration wizard.
  - Click on the Formula editor *button* to open the Formula editor dialog.

- **Open folder**

 © Metastorm Inc., 2008

In this field, you can specify a folder to be opened when the button is pressed. This folder must be identified by its folder ID. You may create and edit the formula using either of the following tools:

- Click on the Integration wizard button to open the Integration wizard
- Click on the Formula editor button to open the Formula editor dialog.

- **New folder action**

In this field, enter the name of the starting action you wish to use to create a new folder. You may enter the name of any starting action that displays a form (e.g. a user action). When the Command button is clicked, the form associated with the specified action will be displayed and a new folder will be created. You must also enter the **New folder map** or the name of an admin form.

- **New folder map**

In this field enter the name of the map that contains the specified **New folder action** or the administration form group for admin forms.

- **Client extensions**

In this field, enter any commands the system must perform when the button is pressed. You may create and edit the formula using either of the following tools:

- Click on the Integration wizard button to open the Integration wizard.
- Click on the Formula editor button to open the Formula editor dialog.

Defining a Check Field

This field is displayed as a check box. Use a check field to display a check value (*TRUE* when checked or *FALSE* when unchecked).

To add a check box, click on the **Check** button on the Form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a check box on the form and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining check field properties:

- Check
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

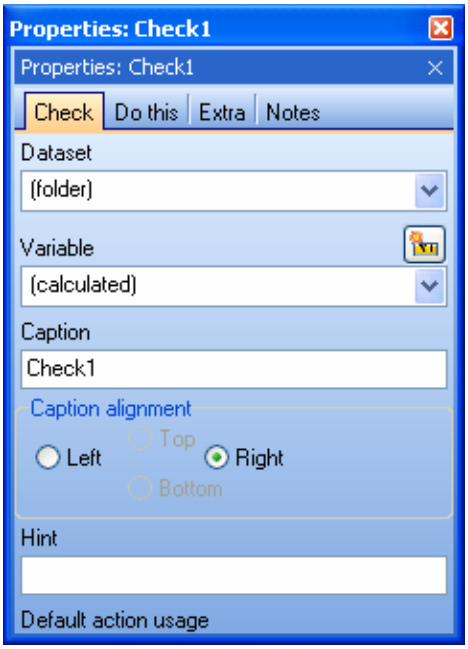


Figure 100: Check Tab

Defining a Check Field's Check *Properties*

1. You can include a text string that will display a brief help message to the user when the cursor hovers over the check box in the client. Type the message you want displayed in the **Hint** field.
2. Identify the variable you wish this field to use in the **Variable** field:

   - If the check box is to be calculated at run-time, select calculated from the list. You will be able to specify how it is calculated on the *Do this* tab.
   - You can create a new custom check variable by clicking on the **New custom variable** button and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this check field.
   - If you are using a dataset, select the column name the system should use to retrieve the information for this field from the column names shown in the drop-down list.

   The variable you select on this tab of the Properties editor will determine the fields that appear on the check field's *Do this* tab.
3. If you are using a variable other than *(calculated)*, Designer uses the variable name as the default caption for the check field. If you wish to change the caption, enter the new caption in the **Caption** field. If you do not provide a caption, or have selected *(calculated)* as your variable, the check field will be displayed with a default caption of "Check" plus an auto-incremented number.
4. The **Dataset** field allows you to select a dataset for the system to use with this field. The default option is *(folder)*, which indicates that you will be using the current folder as the source of data for this field and not a dataset.

 © Metastorm Inc., 2008

Using a dataset means that the source of data for this field is defined in the dataset you selected. Any datasets previously defined for use with this form will be displayed in the drop-down list.

For more information on creating datasets, refer to the Defining a Dataset Field topic in this section.

5. The **Caption alignment** field contains a radio group listing the following alignment position options:

   - **Left** – Places the caption to the left of the check box
   - **Right** – Places the caption to the right of the check box

   Select the caption alignment desired.

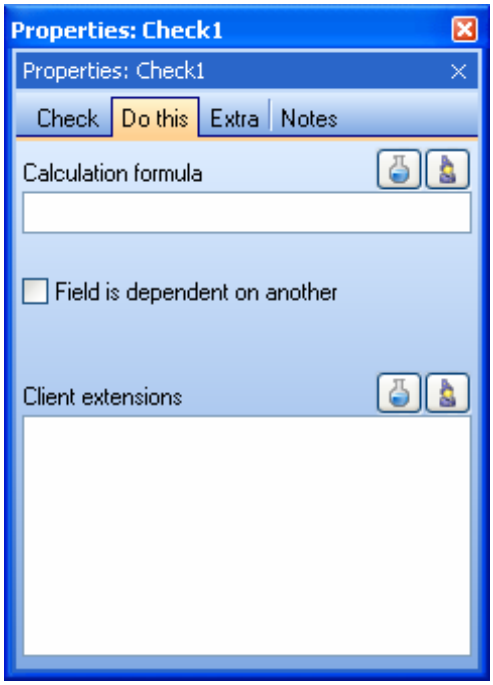


Figure 101: Check / Do this Tab

6. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional** or **hidden** —for the Check field. Use the down-arrow associated with this field to select the desired default.

Defining a Check Field's Do This Properties

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the user changes the value in the check box.

1. If the check field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *page.*

2. If you selected a system variable, or created a new custom variable for this check field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system immediately to notify the Metastorm Engine of any change to this check field. This is necessary if any other fields on the form are dependent upon the content of the check field.

3. The **Calculation formula** field is only available if the variable was set to "Calculated" on the Check tab. This field calculates whether the value of the check field is TRUE (checked) or FALSE (unchecked). You may create and edit the formula using either of the following tools.

    - Click on the Integration wizard button to open the Integration wizard
    - Click on the Formula editor button to open the Formula editor dialog.

4. The **When changed** field appears only if a variable was assigned on the **Check** tab. In this field, enter any command the system is to perform when the contents of this check box are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

5. In the **Client extensions** field, enter any commands to be performed by the system when the focus is given to the checkbox and when focus is removed. You may create and edit the formula using either of the following tools:

    - Click on the Integration wizard button to open the Integration wizard.
    - Click on the Formula editor button to open the Formula editor dialog.

Defining a Currency Field

Values are entered in a currency field as numbers and will be displayed in the specified currency format when the user tabs out of the field. Values will be rounded according to "Bankers Rounding."

Using Bankers Rounding, values below 0.5 are rounded down and values above 0.5 are rounded up. Values of exactly 0.5 are rounded to the nearest even number.

The following numbers are rounded to two decimal places using Bankers Rounding:

- *10.874 becomes 10.87*
- *10.875 becomes 10.88*
- *10.876 becomes 10.88*
- *10.884 becomes 10.88*
- *10.885 becomes 10.88*
- *10.886 becomes 10.89*

To add a currency field, click on the **Currency** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a currency field on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining currency field properties:

- Currency

- Options
- Do this
- Extra
- Notes

For details of the Extra tab and the Notes tab, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Currency Field's Currency *Properties*

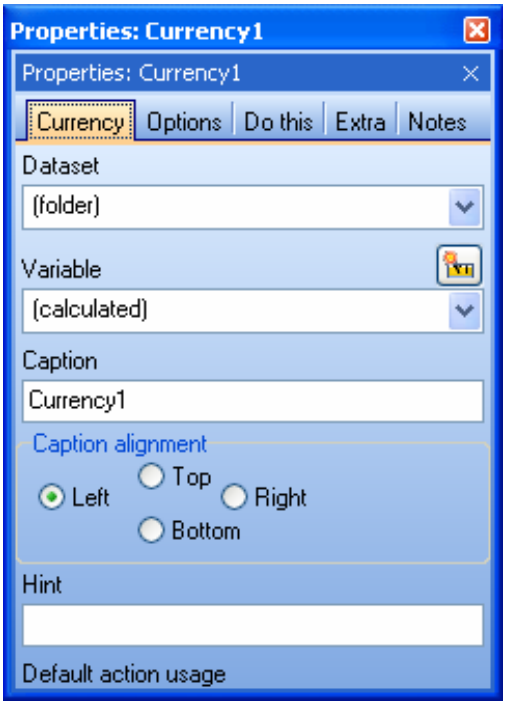


Figure 102: Currency Tab

1. The **Dataset** field allows you to select a dataset for the system to use with this field. The default option is (folder), which indicates that you will be using the current folder as the source of data for this field and not a dataset.
2. If you associate this field with a variable, Designer uses the variable name as the default caption for the currency field. If you wish to change the caption, enter the new caption in the **Caption** field. If you do not provide a caption, or have selected *(calculated)* as your variable, the currency field will be displayed with a default caption of "Currency" plus an auto-incremented number.
3. Identify the variable you wish this field to use in the **Variable** field:
   - If the currency field is to be calculated at run-time, select calculated from the list. You will be able to specify how it is calculated on the *Do this* tab.
   - You can create a new custom currency variable by clicking on the **New custom variable** button and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this currency field.

- If you are using a dataset, select the column name the system should use to retrieve the information for this field from the column names shown in the drop-down list.

The variable you select on this tab of the Properties editor will determine the fields that appear on the currency field's **Do this** tab.

4. The **Caption alignment** field contains a radio group listing the following alignment position options:

   - **Left** – Places the caption to the left of the currency field.
   - **Right** – Places the caption to the right of the currency field.
   - **Top**– Places the caption to the top of the currency field.
   - **Bottom**– Places the caption to the bottom of the currency field.

   Select the caption alignment desired.
5. You can include a text string that will display a brief help message to the user when the cursor hovers over the currency field in the client. Type the message you want displayed in the **Hint** field.

6. The **Default action usage** field allows you to specify a default—such as read-only, optional, hidden, or required—for the Currency field. Use the down-arrow associated with this field to select the desired default.

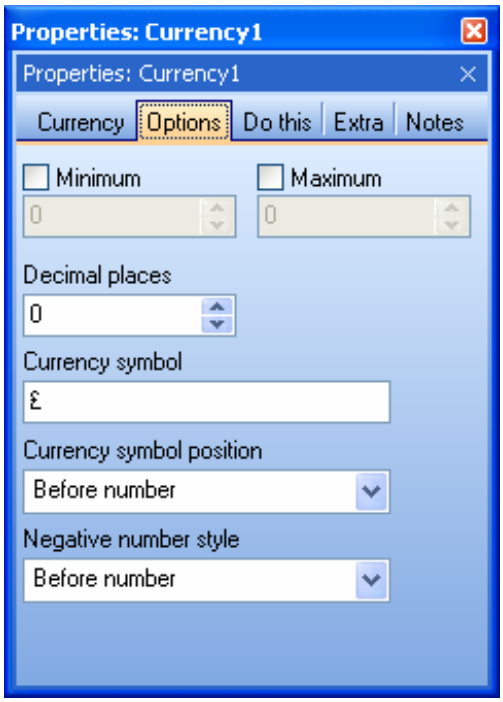


Figure 103: Currency / Options Tab

Defining a Currency Field's Options *Properties*

The *Options* tab allows you to set properties that will determine how the currency value is displayed.

 © Metastorm Inc., 2008

1. If there is a minimum value to be associated with the currency field, click the **Minimum** check box and enter the minimum amount in the field below using the arrows provided.
2. If there is a maximum value to be associated with the currency field, click the **Maximum** check box and enter the maximum amount in the field below using the arrows provided.
3. Set the number of decimal places for the field using the arrows provided in the **Decimal places** field.
4. In the **Currency symbol** field enter the currency symbol for the currency desired.
5. In the **Currency symbol position** field use the drop-down list to select whether the currency symbol will appear before or after the currency number.
6. Use the drop-down list associated with the **Negative number style** to select one of the following options:
    - Placing a minus sign before the negative currency number (default)
    - Placing a minus sign after the negative currency number
    - Placing parenthesis around the negative currency number.

Defining a Currency Field's Do this *Properties*

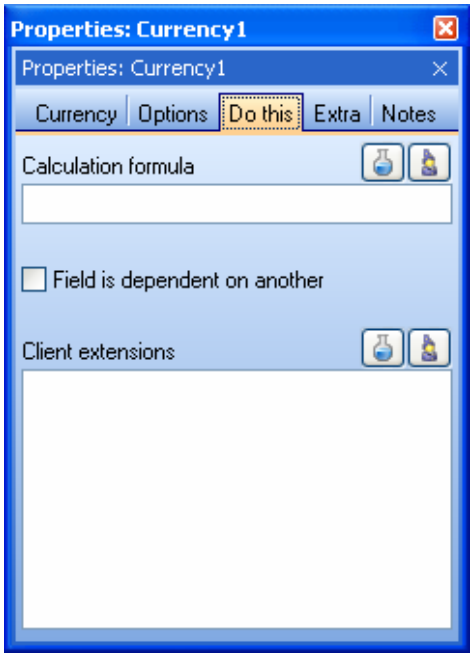


Figure 104: Currency / Do this Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the user changes the value in the currency field.

1. The **Calculation formula** field is available only if the variable was set to "Calculated" on the Currency tab. You may create and edit the formula using either of the following tools.
    - Click on the Integration wizard button to open the Integration wizard.

- Click on the Formula editor button to open the Formula editor dialog.

2. The **When changed** field appears only if a variable was assigned on the *Currency* tab. In this field, enter any command the system is to perform when the value of this currency field is changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.
3. In the **Client extensions** field, enter any commands to be performed by the system when the focus is given to the currency field and when focus is removed. You may create and edit the formula using either of the following tools:
    - Click on the Integration wizard button to open the Integration wizard.
    - Click on the Formula editor button to open the Formula editor dialog.

Defining a Dataset Field

Datasets are used when there are several fields on a form that are to be populated from the same record in a database table. Using a dataset is more efficient than reading the data for each field individually, as the database has only to be read once rather than once for each field.

The Engine will make only one connection to the database for every dataset field.

Each dataset is defined for a specific row of a specific database table. To add a dataset to a form, click on the **Dataset** button on the Form toolbar and then click on the location on the form where you want to place the dataset icon.

Dataset icons are not displayed on the form when displayed through a Metastorm client application – they appear in the Designer window only. However, it may be more convenient to work with other form fields if you place your dataset icons in an out-of-the-way location on your form, such as along a border.

Designer places a dataset icon on the form and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining dataset properties:

- Dataset
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

 © Metastorm Inc., 2008

Defining a Dataset Field's Dataset *Properties*

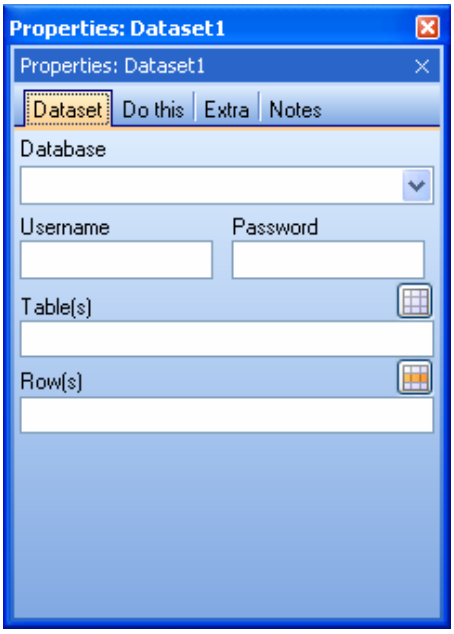


Figure 105: Dataset Tab

1. In the **Database** field, select the database that contains the information you want to use for the dataset. A list of available databases is provided in the drop-down list.

   The data sources available to you should have been created previously. These data sources should be identified in the ODBC Data Source Administrator.

   *If the **Database** field is left blank, the Metastorm database is used.*

2. In the Username and **Password** fields, enter the username and password required to access the database you selected.

3. In the **Table(s)** field, you need to specify which table in the database contains the information you want to use for this dataset. Clicking the **Pick table(s)** button attached to this field will open the **Pick Tables** dialog box.

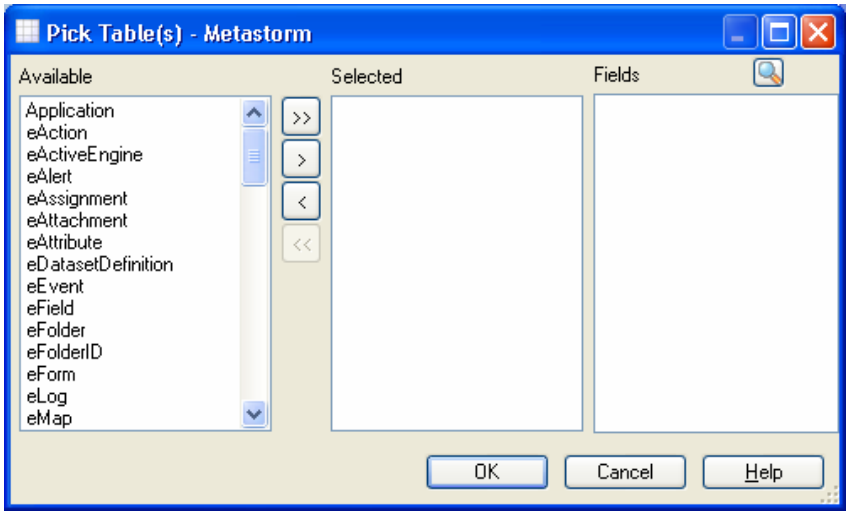


Figure 106: Pick Tables Dialog Box

The **Pick Tables** dialog box identifies the database you are looking at and displays a list of the available tables in the box on the left. Select the table(s) you wish to draw information from and move them into the "selected" list.

You may do this by:

- **Selecting a Table**:

Select a table by double-clicking on the individual table name.

- **Moving the Selected Table**:

Click and drag the selected table into the "selected" field.

Use the > button to move the selected table into the "selected" field.

Designer will automatically search through all tables you select, but will return information only from the first row it encounters that meets the row criteria you specify. While you may select multiple tables, for best results, it is recommended that you select only one table per dataset.

4. Only the first row in the table is loaded. If you want to use multiple rows from the same table, you must specify additional datasets.

You should be very explicit in the criteria you select. Metastorm BPM will load data from the first row that meets your criteria. If your criteria are not specified accurately, you may not call the right row.

If you want to use a row other than the first row of the table, you need to specify the criteria the row should meet in the **Row(s)** field. Clicking the **Pick Row(s)** button attached to the **Row(s)** field opens the **SQL Builder**, which allows you to specify the criteria the desired row must meet.

 © Metastorm Inc., 2008

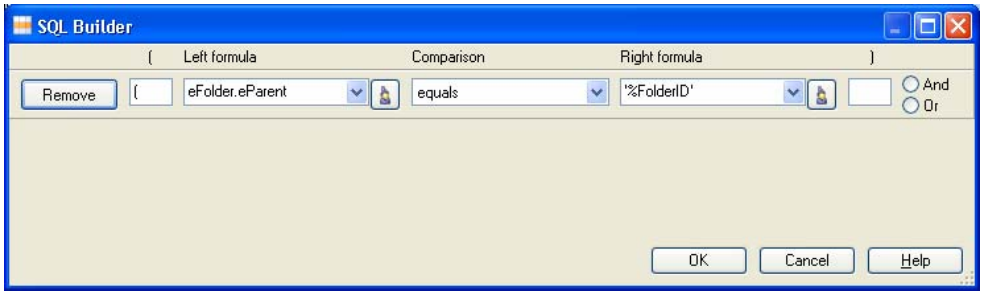


Figure 107: The SQL Builder

When selecting the row, you should be aware of the data types the row contains. If you select a row containing a memo field, you may experience errors, as Designer will interpret any line breaks (hard returns) as new rows, and any tab characters as new columns. If these are present incomplete data will be returned.

This dataset will appear in the Datasets drop-down list that is displayed on other data field Properties editor tabs, and may be used to provide data required by those fields.

If a dataset is associated with a table and columns are added or removed from the table, fields that reference the dataset may show the wrong values. This is because Designer relies on the order and position of the database columns when associating them with form fields. If a table referenced by a dataset does change in this way, you should select the dataset, open the Pick Tables dialog, unselect the table and click OK, and then re-select the table.

Defining a Dataset Field's Do This *Properties*

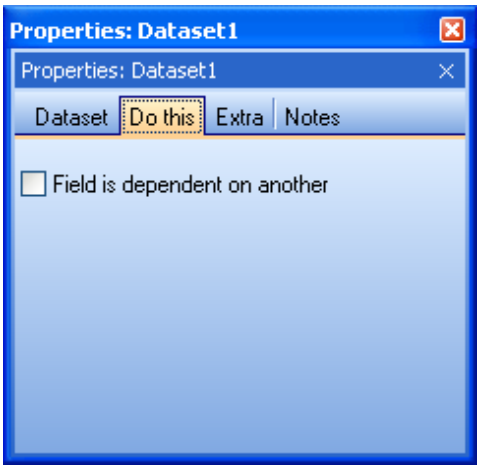


Figure 108: Dataset / Do this Tab

A dataset field's *Do this* tab contains only one property: the **Field is dependent on another** checkbox. Select this option if the information in this dataset may be updated based on changes to another field.

Defining a Date/Time Field

To add a date/time field, click on the **Date/Time** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a date/time field on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining Date/Time field properties:

- Date/Time
- Options
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Date/Time Field's Date/Time *Properties*

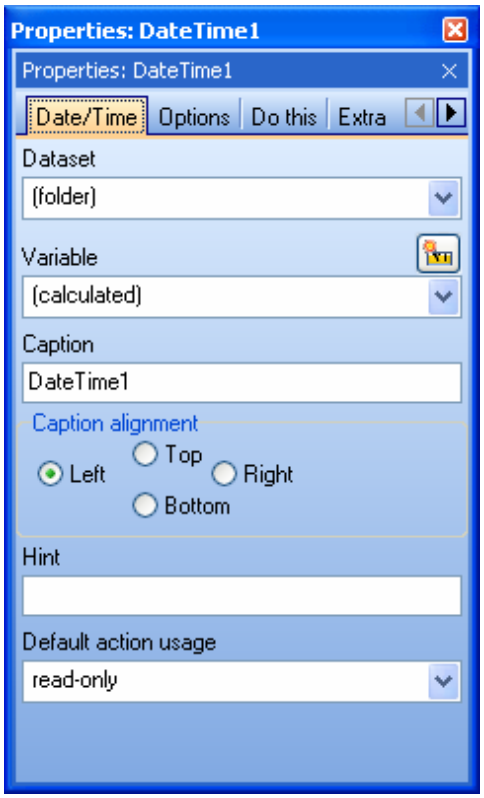


Figure 109: Date/Time Tab

 © Metastorm Inc., 2008

1. The **Dataset** field defines a dataset for the system to use with the time and date field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any information necessary to calculate the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. If you wish to change the caption, enter the new caption in the **Caption** field. If you omit this, or have selected *(calculated)* as your variable, the date/time field will be displayed with a default caption of "DateTime" plus an auto-incremented number.

3. Identify the variable you wish this field to use in the **Variable** field. There are a number of options:

   - If the date is to be calculated at run-time, select *calculated* from the list. You will be able to specify how it is calculated on the *Do this* tab.
   - If the date is to be *stored* in the database, you can select a variable from the drop-down list. Only date/time variables will appear in the list.
   - You can create a new date/time custom variable by clicking on the **New custom variable** button and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this date/time field.
   - If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

   The variable you select on this tab of the Properties editor will determine the fields that appear on the Date/time field's *Do this* tab.

   If you are using a variable other than *(calculated)*, Metastorm BPM uses the variable name as the default caption for the date/time field.

4. The Caption **alignment** field contains a radio group listing following alignment position options:

   - **Left** – Places the caption *to* the left of the date/time field
   - **Right** – Places the caption to the right of the date/time field
   - **Top** – Places the caption *above* the date/time field
   - **Bottom** – Places the caption below the date/time field

   Select the caption alignment desired.

5. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

6. The Default **Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Date/Time field. Use the down-arrow associated with this field to select the desired default.

Defining a Date/Time Field's Options *Properties*

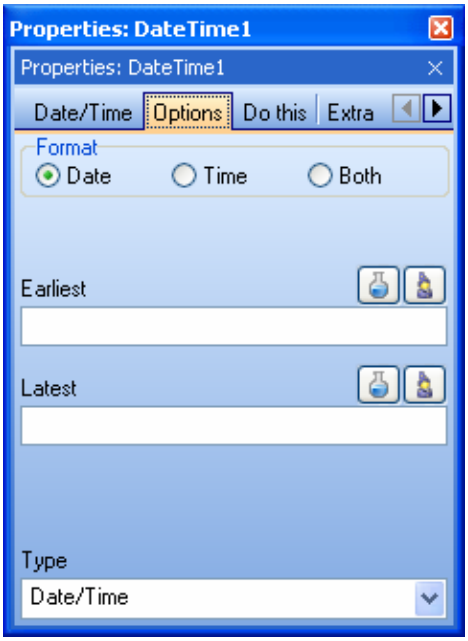


Figure 110: Date/Time / Options Tab

Use the *Options* tab to specify formatting and other options for the date/time field.

1. The Format property contains a radio group allowing you to select from the following options:
   - **Date** – Displays the selected date in the date/time field.
   - **Time** – Displays the selected time in the date/time field.
   - **Both** – Displays the selected date and time together in the date/time field.

   Select the date/time format desired.
2. If you select **Time** or **Both**, the **Disable time zone adjustment** checkbox is available. If you leave this unchecked, the value in the field will be displayed in the client's local time.
3. The **Earliest** field allows you to enter a formula, which will define the earliest valid date or time that may be entered into the date/time field. You may create and edit the formula using either of the following tools:
   - Click on the Integration wizard button to open the Integration wizard.
   - Click on the Formula editor button to open the Formula editor dialog.
4. The **Latest** field allows you to enter a formula, which will define the latest valid date or time that may be entered into the date/time field. You may create and edit the formula using either of the following tools:
   - Click on the Integration wizard button to open the Integration wizard.
   - Click on the Formula editor button to open the Formula editor dialog.

If the date/time field is a required field on the form, you cannot set the Latest *property to be less than or equal to the* Earliest *property, or the user will receive an error when using the form.*

 © Metastorm Inc., 2008

In addition, even though valid maximum and minimum values have been set, if the date/time field is an optional *field on the form, the user may leave the field blank and still have the field pass validation. If the values set in the Properties editor are required minimum/maximum values, the field must be made a* required *field on the form.*

Presetting Date/Time Fields

Designer stores a null date/time as 1/1/1753 00:01.

Process designers should be aware that it is not possible for a user to enter 1/1/1753 00:01 as a valid date/time unless the field has been preset to an alternative date/time.

For example, if the user enters 00:01 into a time field, and submits the form, the next time this form is displayed, the time field will be blank. However, if the field were preset to a date other than 1/1/1753, the field would hold onto the 00:01 time.

In addition, if the field were preset to a date other than 1/1/1753, the value in the time field displayed when the form was first opened would be 00:00. Note that if this field were set to be Required, it would be possible for a user to submit the form without updating the time field.

Defining a Date/Time Field's Do This *Properties*

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the date/time field change.

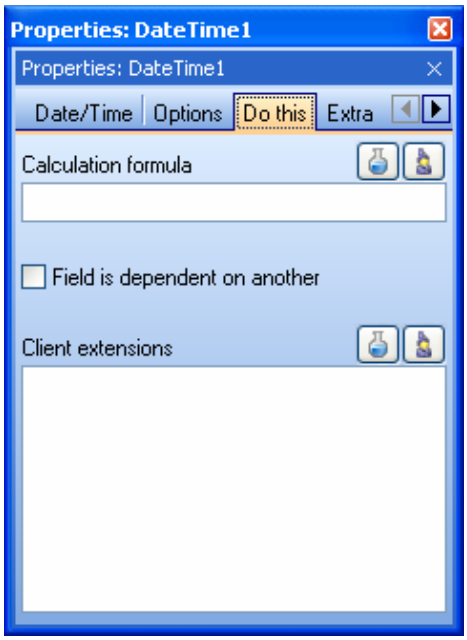


Figure 111: Date/Time / Do this Tab

1. If the date/time field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, check the **Field is dependent on another** check box.

If the field is dependent on another field, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. The **Calculation formula** field allows you to specify how the date/time is calculated. This field is visible only if you selected *(calculated)* as the field's variable. You may create and edit the formula using either of the following tools:

    - Click on the Integration wizard button to open the Integration wizard.
    - Click on the Formula editor button to open the Formula editor dialog.

3. If a custom variable has been assigned to the date/time field, the *Do this* tab will display a **When Changed** option instead of a calculation formula.

    In the **When changed** field, enter any command the system is to perform when the contents of this date/time field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable or created a new custom variable for this date/time field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system to immediately notify the Metastorm engine of any change to this date/time field. This is necessary if any other fields on the form are dependent on this date/time field.

    This box must be checked *if any other fields are dependent upon values in this field.*

5. Enter any commands to be performed by the system when this date/time field gains focus or loses focus in the **Client extensions** field. You may create and edit the formula using either of the following tools:

    - Click on the Integration wizard button to open the Integration wizard.
    - Click on the Formula editor button to open the Formula editor dialog.

Defining a Drop-down Field

This field is displayed as a drop-down list box.

Use a drop-down field to display and select one entry from a list of text or integer values. Alternatively, the list may be generated by a formula that is entered into the *Options list* using the Integration wizard. At run-time the Process Engine will use the formula to determine the contents of the list.

When a formula is used, the items in a drop-down list can be a set of single values or label/value pairs. When label/value pairs are used, the labels are shown in the drop-down and the values are passed to the database.

A static list can be used to create a drop-down list of label/value pairs, for example, the syntax:

```
Label%CHR(9)Value
```

could be used to define:

```
London Gatwick%CHR(9)LGW
Los Angeles%CHR(9)LAX
```

 © Metastorm Inc., 2008

If you need to select multiple entries, you should use a List Field.

To add a drop-down field, click on the **Drop-down** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a drop-down field on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining drop-down field properties:

- Drop-down
- Options
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Drop-down Field's Drop-down *Properties*

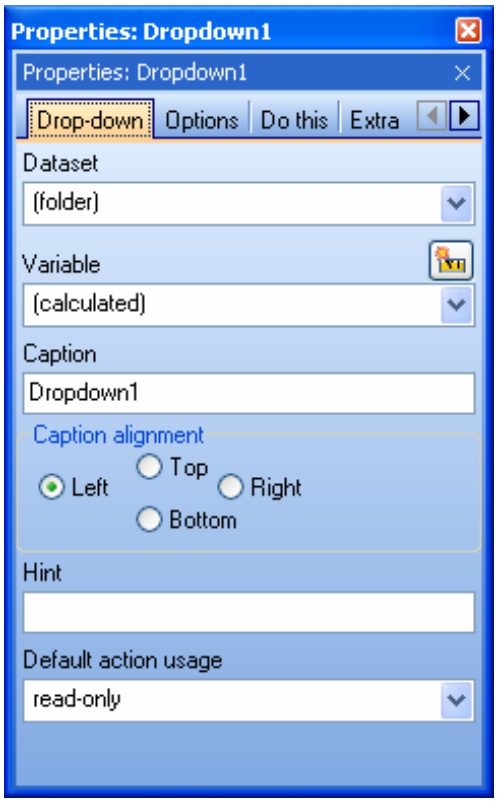


Figure 112: Drop-down Tab

1. In the **Dataset** field enter a dataset for the system to use with the Drop-down list. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. Identify the variable you wish this field to use in the **Variable** field. There are a number of options:

   - If the content of the drop-down list is to be calculated at run-time, select *calculated* from the list. You can specify how it is calculated on the *Do this* tab.
   - If the content of the drop-down list is to be stored in the database, you can select a variable from the drop-down list. Only text or integer variables will appear in the list.
   - You can create a new custom variable by clicking on the **New custom variable** button, , and specifying the details of the variable. The size of the variable must be large enough to hold the largest item that will appear in the drop down list.

   You cannot create a new custom variable if you are using a dataset for this drop-down field.

   - If you are using a dataset, select the column name from the column names shown in the drop-down list.

   The variable you select on this tab of the Properties editor will determine the fields that appear on the drop-down field's *Do this* tab.

   If you are using a variable other than *(calculated)*, Designer uses the variable name as the default caption for the drop-down field.

3. To change the caption, enter the new caption in the **Caption** field. If you omit the caption, or have selected *(calculated)* as your variable, the drop-down field will be displayed with a default caption of "Dropdown" plus an auto-incremented number.

4. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

5. The **Default action usage** field allows you to specify a default status for the Drop-down field from: **read-only**, **optional**, **hidden**, or **required**. Use the down-arrow associated with this field to select the desired default.

   *If the **Variable** field is set to calculated, only the **read-only** and **hidden** options are available.*

6. The **Caption** alignment field contains a radio group listing following alignment position options:

   - **Left** – Places the caption to the left of the drop-down field.
   - **Right** – Places the caption to the right of the drop-down field.
   - **Top** – Places the caption above the drop-down field.
   - **Bottom** – Places the caption below the drop-down field.

   Select the caption alignment desired.

 © Metastorm Inc., 2008

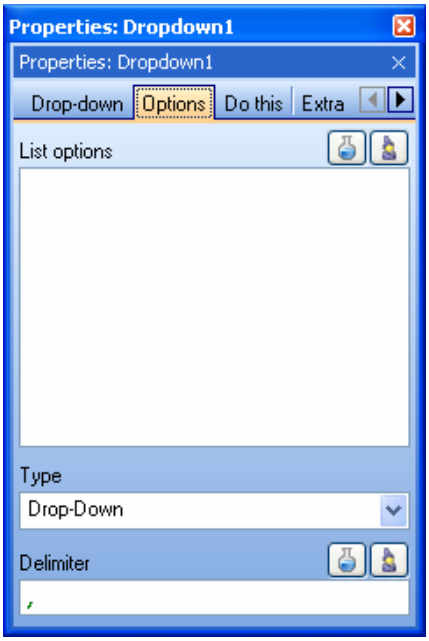


Figure 113: Drop-down / Options Tab

Defining a Drop-down Field's Options *Properties*

Use the *Options* tab to specify the items to appear in the drop-down list. The items in the list are separated by delimiters. The default delimiter is a comma but another character can be used, as defined in the **Delimiter** field.

1. Enter your list items into the **List options** field. You may do this in one of two ways:

    - Open the Integration wizard and choose variables to populate the drop-down list.

    For example, if you select the option *List Registered Users* from the Integration wizard, a list of all users registered with Metastorm BPM will be displayed in the drop-down list. If you want the list to be built from a column or columns of a database table, you would need to select a function such as *List Metastorm data* (for single values) or *List Metastorm label/value pair* (for label/value pairs) from the Integration wizard options.

    Items in the list are separated by delimiters. The default delimiter is a comma; an alternative delimiter can be specified, as set out in the *Delimiters* section below.

For more information refer to Section 10, Creating Formulas.

    - Type the list items directly into the field.

    This is a simple method to use when you have a relatively short list. Either type the items directly or click on the Formula editor button and type the list items into the Formula editor field, separating items with a comma or an alternative delimiter, see Delimiter field below.

    You may mix typed options with formulas. For example, you may type the first line as "Other" and define the second line to be a formula that extracts a list from the database table.

    These values will be evaluated when the form is loaded or, if the field is specified as dependent, whenever any field specified as having dependants is changed.

To use negative numbers, parentheses and percentage signs in a dropdown list, the following should be taken into account:

- Any item in a drop-down list containing negative numbers or parentheses without evaluation expressions can be written as they are displayed. For example, -1 displays as -1; London (Heathrow) displays as London (Heathrow).
- Any item in a drop-down list containing negative numbers or parentheses with quotation marks in the dropdown list are written as they are displayed. For example, "-1" is displayed as "-1"; "London (Heathrow)" is displayed as "London (Heathrow)".
- If any item in a drop-down list contains negative numbers or parentheses with expressions, the item must contain the character codes for hyphens and parenthesis. For example, **Book a flight to London %CHR(41)Heathrow%CHR(41) by %deadline** displays **Book a flight to London (Heathrow) by 28/09/2007** where deadline=28th September 2007. **%Priority CHR(45)1** displays **9-1** where priority=9.
- If an item in a dropdown list contains a percentage character, another percentage character must be added in order for the first to be displayed. For example, the item **100%%** will be displayed as **100%**.

2. Delimiters that separate items in the drop-down list are entered into the **Delimiters** field.

If a comma is defined as the delimiter between items in a drop-down list and the value read from a database is London, England, then this value will appear as two separate items on two separate lines in the dropdown list. If the delimiter is '|' then the entry 'London, England|' will appear as one item, 'London, England', on one line of the drop-down list.

*If the list contains a Metastorm BPM expression, for example, %GetData or %SelectSQL, which uses a delimiter other than that specified in the **Delimiter** field, the delimiter specified in the expression is used.*
For example, the expression
`%GetData (DBName, DBTable, Rows, Column,, @)`
specifies that @ should be used as a row delimiter.
However, if this expression is enclosed by another Metastorm BPM expression, the delimiter specified in the expression is not used.
For example, if the list contains the expression
`%Replace (%GetData (DBName, DBTable, Rows, Column,, @), "blah", "blah1", 5)`
*the delimiter specified in the **Delimiter** field is used.*

Delimiters can be entered in one of three ways:

a) In the **Delimiter** field type the delimiter character, e.g. | or :

- The default delimiter is a comma which can be deleted. Alternatively, type the delimiter after the comma in which case the new entry overwrites the comma.
- Delimiters must of a type that have Unicode values in the range 0-127 e.g., pipe(|), carriage return (CR) , and tab.
- A delimiter can be entered as a character variable, e.g. %CHR(9) (= Tab).

When the cursor is held over a character variable entry the character type is displayed.

 © Metastorm Inc., 2008

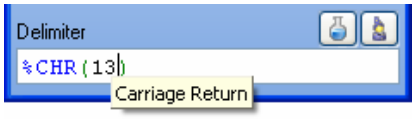


Figure 114: Delimiter description

b) Delimiters can be entered by using the Integration Wizard. To enter a delimiter using the Integration Wizard:

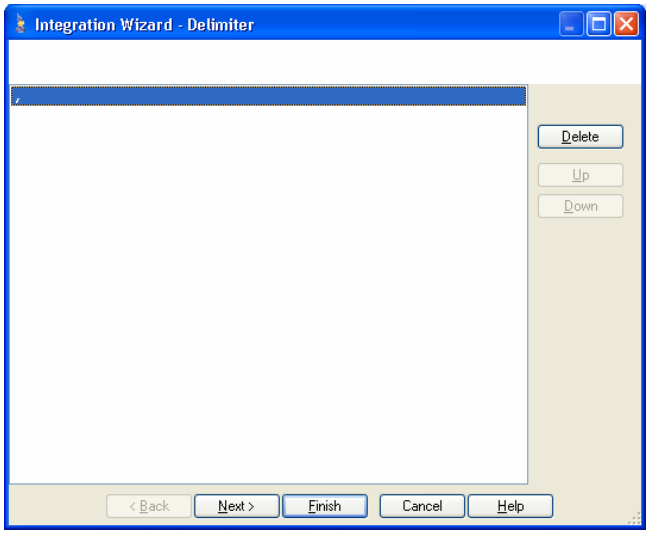


Figure 115: Delimiter defined by Integration Wizard

- Select the Wizard icon above the **Delimiter** field,
  - The Integration Wizard dialog appears;

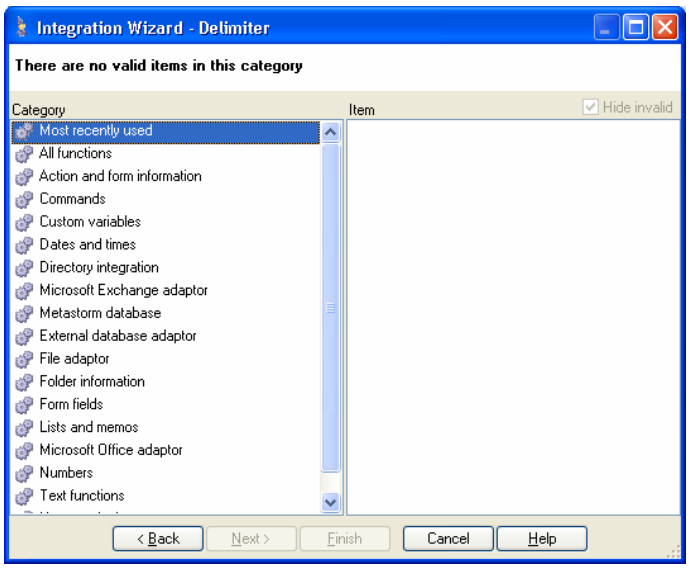


Figure 116: Delimiter Options

- Select **Delete** to remove any existing delimiters or select **Next**;
  - The Wizard displays the Text option from a list of variable categories.

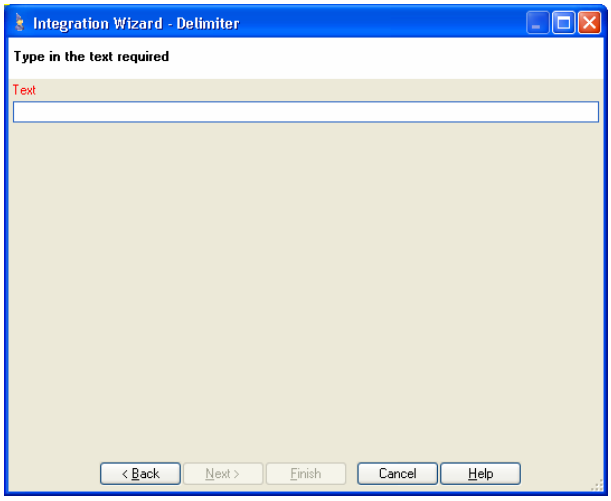


Figure 117: Delimiter text entry

- Select **Next**:
  - The text entry field is displayed.
- Enter the required delimiter and click **Finish;**
  - The delimiter is displayed in the **Delimiter** field of the *Properties* editor.

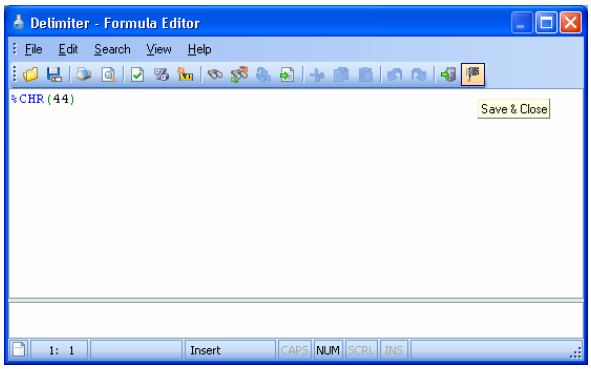


Figure 118: Delimiter - Formula Editor

c) The third way to enter a delimiter is to select the Formula editor icon, , above the **Delimiter** field;

- The Formula Editor is displayed;
- Enter the required delimiter and
- Click the **Save and Close** icon.

If the first entry of the delimiter is' %' a drop down list is presented.

Defining a Drop-down Field's Do This *Properties*

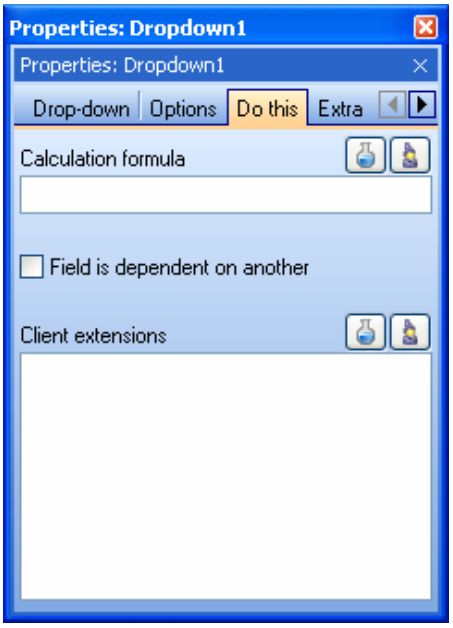


Figure 119: Drop-down / Do this Tab

Use the **Do this** tab to specify the calculation formula, dependencies, and processing to be carried out when items are selected from the drop-down list.

1. If the drop-down field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. This field is visible only if you selected *(calculated)* as the field's variable. You may create and edit the formula using either of the following tools:

   - Click on the Integration wizard button to open the Integration wizard.
   - Click on the Formula editor button to open the Formula editor dialog.

3. If a custom variable has been assigned to the drop-down field, the *Do this* tab will display a **When Changed** option instead of a calculation formula.

   In the **When changed** field, enter any command the system is to perform when the contents of this drop-down field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable, or created a new custom variable for this drop-down field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system to immediately notify the Metastorm engine when any item in this drop-down field is selected. This is necessary if any other fields on the form are dependent upon the content of the drop-down field.

   This box must be checked *if any other fields are dependent upon values in this field.*

5. Enter any commands to be performed by the system when the drop-down field is entered or exited in the **Client extensions** field. You may create and edit the formula using either of the following tools:

   - Click on the Integration wizard button to open the Integration wizard.
   - Click on the Formula editor button to open the Formula editor dialog.

Defining a Default Value for a Drop-down Field

A drop-down list can have a default value. A default value is setup in the **Do This** tab of a Form Properties.

In the field **When user loads form** set the variable name equal to the default value. For example using a static value::

```
%VariableName:=”Value”
```

Default Values can also be setup using a formula.

Defining a List Field

Use a list field to display and select multiple entries from a list of text values. The list may be generated by a formula specified via the Integration wizard or Formula editor, and calculated by the Engine at run-time.

When a formula is used, the items in a list field can be a set of single values or label/value pairs. When label/value pairs are used, the labels are shown in the list and the values are passed to the database.

To add a list field, click on the **List** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a list field on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining list field properties:

- List
- Options
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a List Field's List *Properties*

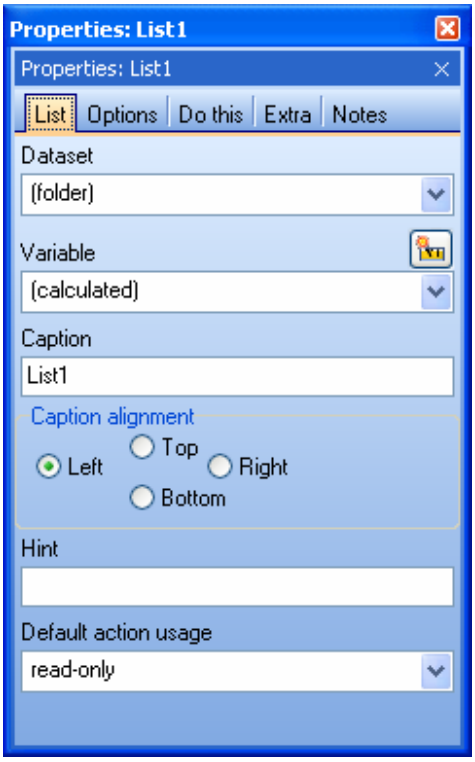


Figure 120: List Tab

1. The **Dataset** field asks you to define a dataset for the system to use with this field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. If you wish to change the caption, enter the new caption in the **Caption** field. If you omit this, or have selected *(calculated)* as your variable, the list field will be displayed with a default caption of "List" plus an auto-incremented number.

3. Identify the variable you wish this field to use in the **Variable** field. There are a number of options:

   - If items in the list are to be selected at run-time, select *calculated*. You will be able to specify how it is calculated on the *Do this* tab.
   - If the list field will use any system or previously defined custom memo variables, you can select a variable from the drop-down list. Only memo variables will appear in the list.
   - You can create a new memo custom variable by clicking on the **New custom variable** button and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this list field.
   - If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

   The variable you select on this tab of the Properties editor will determine the fields that appear on the list field's *Do this* tab.

   If you are using a variable other than *(*calculated*)*, Designer uses the variable name as the default caption for the list field.

4. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

5. The **Caption** alignment field contains a radio group listing following alignment position options:

   - **Left** – Places the caption to the left of the list field.
   - **Right** – Places the caption to the right of the list field.
   - **Top** – Places the caption above the list field.
   - **Bottom** – Places the caption below the list field.

   Select the caption alignment desired.

6. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the List field. Use the down-arrow associated with this field to select the desired default.

 © Metastorm Inc., 2008

Defining a List Field's Options *Properties*

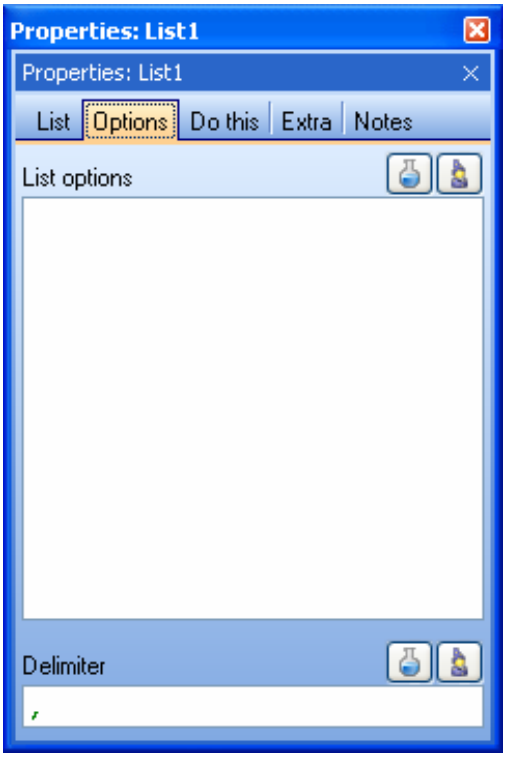


Figure 121: List / Options Tab

Use the *Options* tab to define the items contained in the list field.

1. Enter your list items into the **List options** field. You may do this in one of two ways:

   - Open the Integration wizard and choose variables to populate the list.

     For example, if you select the option *List Registered Users* from the Integration wizard, a list of all users registered with Metastorm BPM will be displayed in the list. If you want the list to be built from a column or columns of a database table, you would need to select a function such as *List Metastorm data* (for single values) or *List Metastorm label/value pair* (for label/value pairs) from the Integration wizard options.

   - Type the list items directly into the field.

     This is a simple method to use when you have a relatively short list. Either type the items directly into the List Options property, one per line, or click on the **Formula editor** button and type the list items into the Formula editor field. Press **Enter** after each option.

   These values will be evaluated when the form is loaded or, if the field is specified as dependent, whenever any field specified as having dependants is changed.

   To use negative numbers, parentheses and percentage signs in a list, the following should be taken into account:

   - If the first item in the list is a negative number, enclose it in quotation marks. For example, **"-1"** will be displayed as **-1**. Subsequent negative numbers in the list should not be enclosed in quotation marks.

- If the first item in the list contains one or more parentheses, the item should be enclosed in quotation marks. For example, **"London (Heathrow)"** will be displayed as **London (Heathrow)**. If this item should be displayed with quotes, use **%"** to represent a displayed quote. For example, **"%"London (Heathrow)%""** will be displayed as "**London (Heathrow)"**. Subsequent items containing parentheses should not be enclosed in quotation marks.
- If an item in a list contains a percentage character, another percentage character must be added in order for the first to be displayed. For example, the item **100%%** will be displayed as **100%**.

2. Delimiters that separate items in the drop-down list are entered into the **Delimiters** field.

If a comma is defined as the delimiter between items in a drop-down list and the value read from a database is London, England, then this value will appear as two separate items on two separate lines in the list. If the delimiter is' |' then the entry London, England| will appear as one item, London, England, on one line of the list.

Delimiters can be entered in one of three ways, refer to *Delimiters* in Defining a Drop-down List's Properties, *page 173* for details of how to enter delimiters.

Defining a List Field's Do This *Properties*

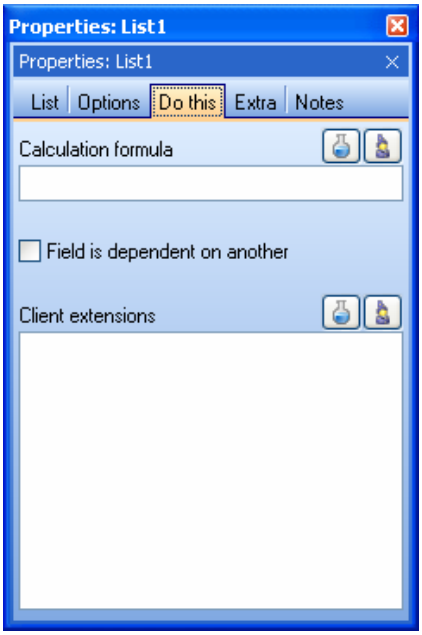


Figure 122: List / Do this Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out when an item is selected from a list field.

1. If the list field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its Do this tab.*

 © Metastorm Inc., 2008

2. The **Calculation formula** field is visible only if you selected *(calculated)* as the field's variable. You may create and edit the formula using either of the following tools:

    - Click on the Integration wizard button to open the **Integration wizard**.
    - Click on the Formula editor button to open the Formula **editor** dialog.

3. If a custom variable has been assigned to the list field, the *Do this* tab will display a **When Changed** option instead of a calculation formula.

In the **When changed** field, enter any command the system is to perform when the contents of this list field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable, or created a new custom variable for this list field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system to immediately notify the Metastorm Engine of any change to this list field. This is necessary if any other fields on the form are dependent upon the content of the list field.

This box must be checked *if any other fields are dependent upon values in this field.*

5. Enter any commands to be performed by the system when this list field gains focus or loses focus in the **Client extensions** field. You may create and edit the formula using either of the following tools:

    - Click on the Integration wizard button to open the Integration wizard.
    - Click on the Formula editor button to open the Formula editor dialog.

Defining a Memo Field

Use a memo field to display or edit a memo value.

Process designers should be aware that if memo fields are to be used for holding XML data, limitations exist. The character sequence ']]>' will result in an error. To avoid this error, a client side script should be used to convert this sequence to ']]]>'.

To add a memo field, click on the **Memo** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a memo field on the form and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining memo field properties:

- Memo.
- Do this.
- Extra.
- Notes.

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Memo Field's Memo *Properties*

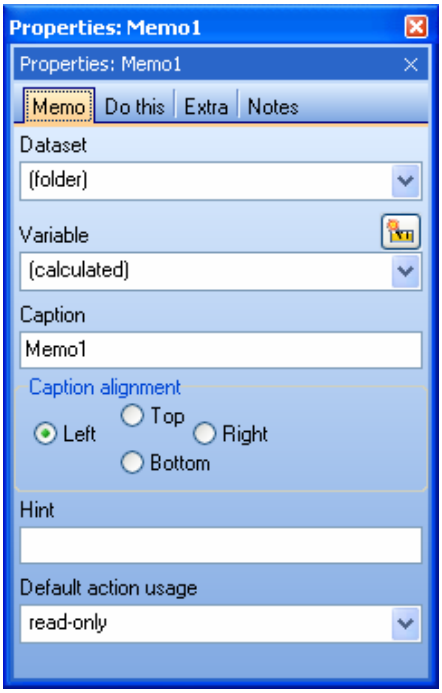


Figure 123: Memo Tab

1. The **Dataset** field asks you to define a dataset for the system to use with this field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset field* topic in this section.

2. If you wish to change the caption, enter the new caption in the **Caption** field. If you choose not to define a caption, and selected *(calculated)* as your variable, the memo field will be displayed with a default caption of "Memo" plus an auto-incremented number.

3. Identify the *variable* you wish this field to use in the **Variable** field. There are a number of options:

   - If the memo is to be calculated at run-time, select calculated from the list. You will be able to specify how it is calculated on the *Do this* tab.
   - If the memo is to be stored in the database, you can select a variable from the drop-down list. Only memo variables will appear in the list.
   - You can create a new memo custom variable by clicking on the **New custom variable** button and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this memo field.
   - If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

 © Metastorm Inc., 2008

The variable you select on this tab of the Properties editor will determine the fields that appear on the memo field's *Do this* tab.

If you are using a variable other than *(calculated)*, Designer uses the variable name as the default caption for the memo field.

4. The **Caption alignment** field contains a radio group listing following alignment position options:

   - **Left** – Places the caption to the left of the memo field.
   - **Right** – Places the caption to the right of the memo field.
   - **Top** – Places the caption above the memo field.
   - **Bottom** – Places the caption below the memo field.

   Select the caption alignment desired.

5. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

6. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Memo field. Use the down-arrow associated with this field to select the desired default.

Defining a Memo Field's Do This *Properties*

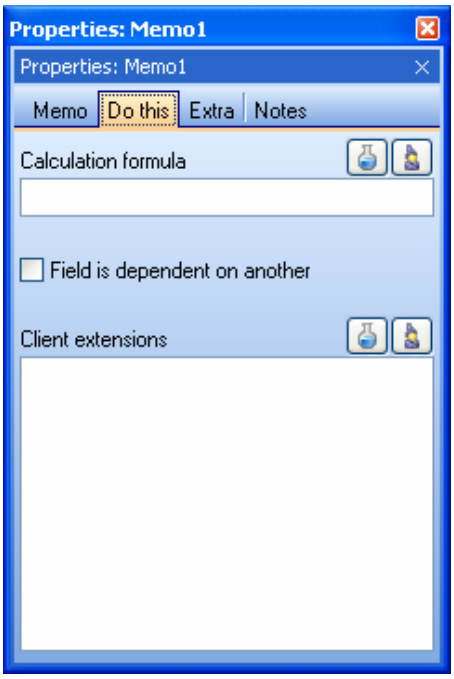


Figure 124: Memo / Do this Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the memo field change.

1. If the memo field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. The **Calculation formula** field allows you to specify how the contents of the memo field are calculated. This field is visible only if you selected *(calculated)* as the field's variable. You may create and edit the formula using either of the following tools:
   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.
3. If a custom variable has been assigned to the memo field, the *Do this* tab will display a **When Changed** option instead of a calculation formula.

   In the **When changed** field, enter any command the system is to perform when the contents of this memo field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable, or created a new custom variable for this memo field, the **Field has dependants of its own** check box will be displayed on this page of the Properties editor. This field instructs the system to notify immediately the Metastorm Engine of any changes to this memo field. This is necessary if any other fields on the form are dependent upon the content of the memo field.

   This box must be checked *if any other fields are dependent upon values in this field.*

5. Enter any commands to be performed by the system when the memo field is entered or exited in the **Client extensions** field. You may create and edit the formula using either of the following tools:
   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.

Defining a Number Field

The number field is displayed as a single-line field that accepts only numeric input. This input may be supplied by a user or may be supplied by the database based on variables you specify. Use a number field to display an integer (whole) or real (decimal) value.

Variables of type Real offer limited precision. Use a variable of type Currency to guarantee the most precise calculations.

Note that due to a JavaScript limitation, any number above 29,999,999,999,999.5070 containing decimal places, which is entered into a number or currency field in the Web client will be rounded. For example, 39,999,999,999,999.5070 will become 39,999,999,999,999.5100.

This should be taken into consideration when designing forms.

To add a number field, click on the **Number** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

 © Metastorm Inc., 2008

Designer places a number field on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining number field properties:

- Number
- Options
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Number Field's Number *Properties*

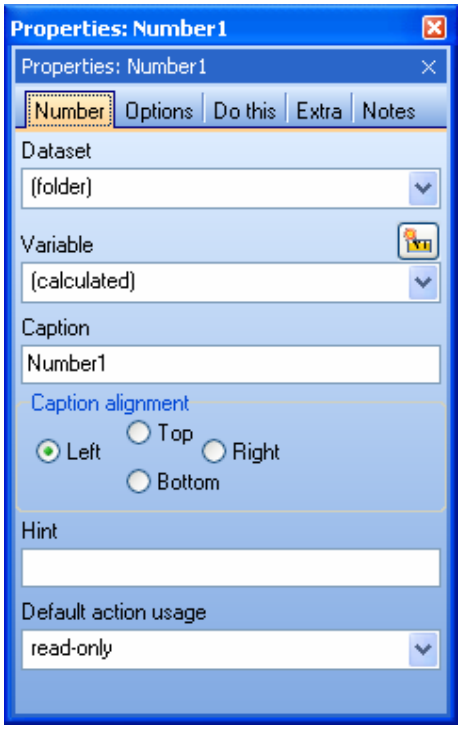


Figure 125: Number Tab

1. The **Dataset** field asks you to define a dataset for the system to use with this field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. Enter the variable you wish this field to use in the **Variable** field. There are a number of options:

- If the number is to be calculated at run-time, select *calculated* from the list. You will be able to specify how it is calculated on the *Do this* tab.
- If the number is to be stored in the database, you can select a variable from the drop-down list. Only numeric variables will appear in the list.
- You can create a new custom integer or real variable by clicking on the **New custom variable button** and specifying the details of the variable. When you add a custom variable, you must select whether the field will hold integer data or decimal numbers (real). You cannot create a new custom variable if you are using a dataset for this number field.
- If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

The variable you select on this tab of the Properties editor will determine the fields that appear on the Number field's *Do this* tab.

3. If you are using a variable other than *(calculated)*, Designer uses the variable name as the default caption for the number field. If you wish to change the caption, enter the new caption in the **Caption** field. If you omit this, or have selected *(calculated)* as your variable, the number field will be displayed with a default caption of "Number" plus an auto-incremented number.

4. The **Caption alignment** field contains a radio group listing following alignment positions:

   - **Left** – Places the caption to the left of the number field.
   - **Right** – Places the caption to the right of the number field.
   - **Top** – Places the caption above the number field.
   - **Bottom** – Places the caption below the number field.

   Select the caption alignment desired.

5. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

6. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Number field. Use the down-arrow associated with this field to select the desired default.

 © Metastorm Inc., 2008

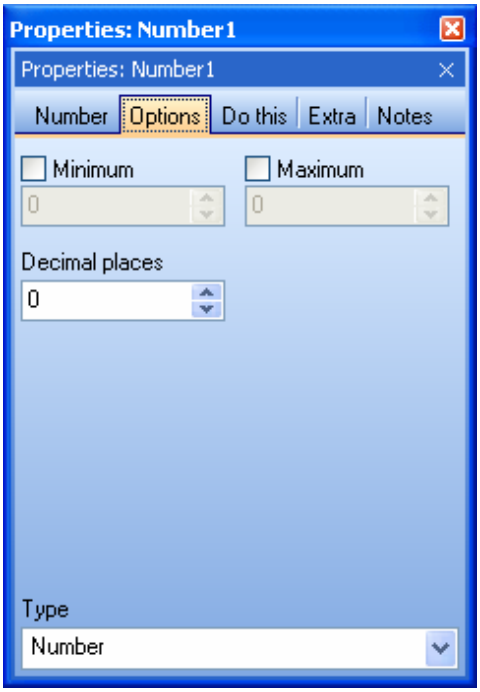


Figure 126: Number / Options Tab

Defining a Number Field's Options *Properties*

For a number field, you can specify a maximum and minimum value for the field and, for a number field associated with a real variable, the number of decimal places.

1. To set a minimum size for the number field, check the box marked **Minimum**.

   In the **Minimum** field, specify the smallest value that may be entered into the number field.

2. To set a maximum size for the number field, check the box marked **Maximum**.

   In the **Maximum** field, specify the largest value that may be entered into the number field.

   You cannot set a Maximum *value lower than a* Minimum *value. For example, if you set the* Minimum *option to 10, the lowest possible* Maximum *value will automatically be set to 10.*

   In addition, even though valid maximum and minimum values have been set, if the number field is an optional *field on the form, the user may leave the field blank and still have the field pass validation. If the values set in the Properties editor are required minimum and maximum values, the field must be made a* required *field on the form.*

3. In the **Decimal places** field specify the number of decimal places the field should display. This field will be unavailable (grayed out) if the number field is associated with an integer variable.

4. The field type is displayed in the **Type** drop-down list. If you wish to change the type of field you are working with, select the new type from the list.

   If you change the field type after you have defined the field's properties, be sure to review the properties for the new field type.

Defining a Number Field's Do This *Properties*

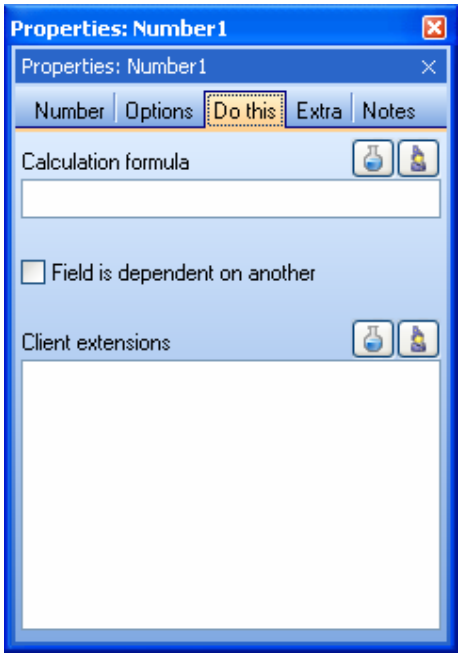


Figure 127: Number / Do this Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the number field change.

1. If the number field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. The **Calculation formula** calculates the number that will appear in the number field. You may create and edit the formula using either of the following tools:
   - Click on the Integration **wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.
3. If a custom variable has been assigned to the number field, the **Do this** tab will display a **When Changed** option instead of a calculation formula.

   In the **When changed** field, enter any command the system is to perform when the contents of this list field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.
4. Enter any commands to be performed by the system when the number field is entered or exited in the **Client extensions** field. You may create and edit the formula using either of the following tools:
   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.

 © Metastorm Inc., 2008

5. If you selected a system variable, or created a new custom variable for this number field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system to notify immediately the Metastorm Engine of any change to this number field. This is necessary if any other fields on the form are dependent upon the content of the number field.

Defining a Radio Group Field

Radio group values must be pre-defined as a list. They cannot be determined by the Engine when the form is loaded.

If you need the contents of the list to vary dynamically, you should use a drop-down field.
If you need to be able to select multiple entries, you should use a list field.

To add a radio group, click on the **Radio group** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a radio group on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining radio group properties:

- Radio group
- Options
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

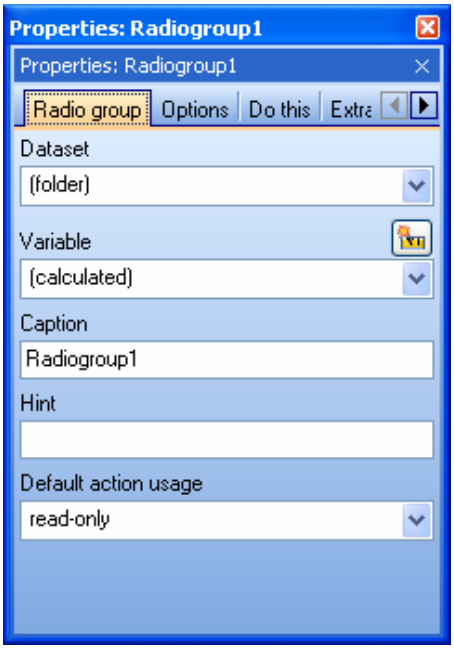


Figure 128: Radio group Tab

Defining a Radio Group Field's Radio Group *Properties*

1. The **Dataset** field asks you to define a dataset for the system to use with this field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. Identify the variable you wish this field to use in the **Variable** field. There are a number of *options*:

   - If the text is to be calculated at run-time, select *calculated* from the list. You will be able to specify how it is calculated on the *Do this* tab.
   - If the text is to be stored in the database, you can select a variable from the drop-down list.
   - You can create a new text or integer custom variable by clicking on the **New custom variable button** and specifying the details of the variable. The size of the custom variable must be at least as long as the length of the largest caption that the radio group will display. You cannot create a new custom variable if you are using a dataset for this radio group field.
   - If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

   The variable you select on this tab of the Properties editor will determine the fields that appear on the radio group's *Do this* tab.

   If you are using a variable other than *(calculated)*, Designer uses the variable name as the default caption for the radio group.

   If the radio group selected has a value longer than the size of the variable the radio group is associated with, the value will be truncated when it is stored in the database. When the radio group is next displayed, there will be no match to any button and it will appear that no button has been selected.

3. If you wish to change the caption, enter the new caption in the **Caption** field. If you omit this, or have selected *(calculated)* as your variable, the radio group will be displayed with a default caption of "Radio group" plus an auto-incremented number.

   Radio group captions are always placed at the top of the radio group field.

4. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

5. The **Default Action Usage** field allows you to specify a default—such as **read-only**, optional, **hidden**, or **required**—for the Radio Group field. Use the down-arrow associated with this field to select the desired default.

 © Metastorm Inc., 2008

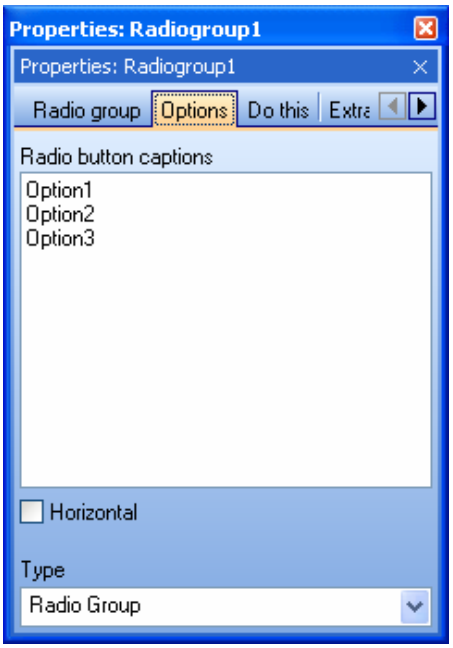


Figure 129: Radio group / Options Tab

Defining a Radio Group Field's Options *Properties*

Use the *Options* tab to identify the text and layout options for the radio group. The default layout displays three option buttons.

1. Double-click a default option to edit its caption or delete it from the list using your delete key. To add radio buttons type them directly into the **Radio button captions** field, pressing **Enter** after each new option.

   Designer will create a new radio button for each item in the list. For a non-calculated field, the text of the selected option will be stored in the database.

2. Checking the **Horizontal** option instructs Designer to align the radio group buttons horizontally across the form. If you leave this box unchecked, the radio group buttons will be aligned vertically.

3. The field type is displayed in the **Type** drop-down list. If you wish to change the type of field you are working with, select the new type from the list.

If you change the field type after you have defined the field's properties, be sure to review the properties for the new field type.

Defining a Radio Group Field's Do This *Properties*

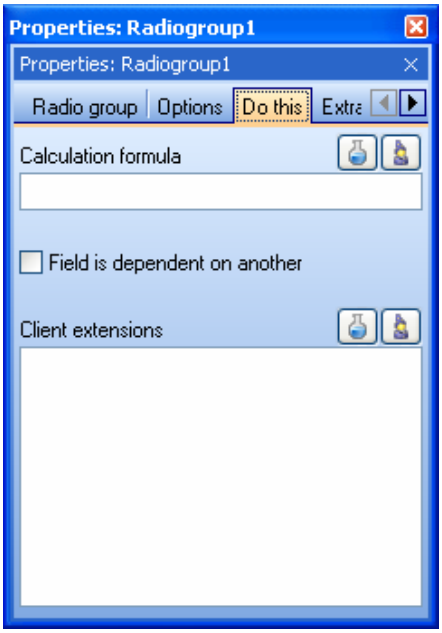


Figure 130: Radio group / Options Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the radio group change.

1. If the radio group may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. The **Calculation formula** field allows you to specify how the contents of the radio group are calculated. This field is visible only if you selected *(calculated)* as the field's variable. You may create and edit the formula using either of the following tools:
   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.
3. If a custom variable has been assigned to the radio group, the **Do this** tab will display a **When Changed** option instead of a calculation formula.

   In the **When changed** field, enter any command the system is to perform when the contents of this radio group are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable, or created a new custom variable for this radio group, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system immediately to notify the Metastorm Engine when one of this radio group's options is selected. This is necessary if any other fields on the form are dependent upon the content of the radio group.

 © Metastorm Inc., 2008

This box must be checked if any other fields are dependent upon values in this field.

5. Enter any commands to be performed by the system when this radio group gains focus or loses focus in the **Client extensions** field. You may create and edit the formula using either of the following tools:

    - Click on the **Integration wizard** button to open the Integration wizard.
    - Click on the **Formula editor** button to open the Formula editor dialog.

Defining a Signature Field

Use a signature field to add a signature to a form. Users completing the form may (or must, if the field is required) provide an electronic signature, which will be recorded in the Metastorm database.

In order for a signature field to incorporate the current user's ID or user name, a text field called "UserID" must exist on the form but this can be assigned to any variable or be calculated. When the signature script runs, the value of the UserID field is included in the signature; otherwise the text Metastorm user is added.

In addition, the procedure designer must assign the logged on user's name to the UserID *field.*

To add a signature field, click on the **Signature** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a signature field on the form and displays the corresponding Properties editor.

The Properties editor displays four tabs for defining signature field properties:

- Signature
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Signature Field's Signature *Properties*

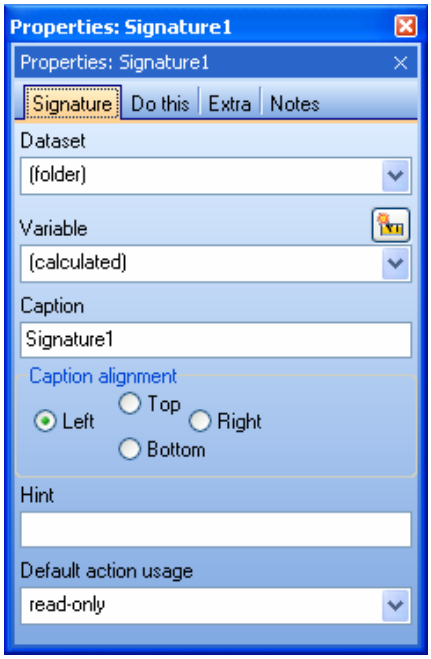


Figure 131: Signature Tab

1. The **Dataset** field defines a dataset *for* the system to use with the signature field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. If you wish to change the caption, enter the new caption in the **Caption** field. If you omit this, or have selected *(calculated)* as your variable, the signature field will be displayed with a default caption of "Signature" plus an auto-incremented number.

3. Identify the variable you wish this field to use in the **Variable** field. There are a number of options:

   - If the signature is to be calculated at run-time, select *calculated* from the list. You will be able to specify how it is calculated on the *Do this* tab.
   - If the signature is to be stored in the database, you can select a variable from the drop-down list. Only memo variables will appear in the list.
   - You can create a new custom signature variable by clicking on the **New custom variable button** and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this signature field.
   - If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

 © Metastorm Inc., 2008

The variable you select on this tab of the Properties editor will determine the fields that appear on the signature field's Do *this* tab.

4. The **Caption** alignment field contains a radio group listing following alignment position options:

   - **Left** – Places the caption to the left of the signature field
   - **Right** – Places the caption to the right of the signature field
   - **Top** – Places the caption above the signature field
   - **Bottom** – Places the caption below the signature field

   Select the caption alignment desired.

5. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

6. The Default **Action Usage field** allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Signature field. Use the down-arrow associated with this field to select the desired default.

Defining a Signature Field's Do this *Properties*

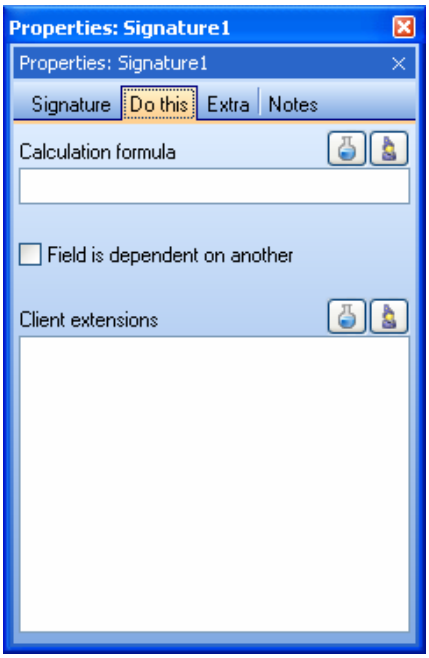


Figure 132: Signature / Do this Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the signature field change.

1. If the signature field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. The **Calculation formula** field allows you to specify how the contents of the signature field are calculated. This field is visible only if you selected *(calculated)* as the field's variable. You may create or edit the formula using either of the following tools:

    - Click on the **Integration wizard** button to open the Integration wizard.
    - Click on the **Formula editor** button to open the Formula editor dialog.

3. If a custom variable has been assigned to the signature field, the **Do this** tab will display a **When changed** option instead of a calculation formula.

    In the **When changed** field, enter any command the system is to perform when the contents of this signature field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable or created a new custom variable for this signature field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system to notify immediately the Process Engine of any change to this signature field. This is necessary if any other fields on the form are dependent upon the content of the signature field.

    This box must be checked *if any other fields are dependent upon values in this field.*

5. Enter any commands to be performed by the system when the signature field gains or loses focus in the **Client extensions** field. You may create and edit the formula using either of the following tools:

    - Click on the **Integration wizard** button to open the Integration wizard.
    - Click on the **Formula editor** button to open the Formula editor dialog.

Defining a Status Field

You can place a status field on any form. A status field defines a value in one of three states; Safety, Warning or Danger. The state depends on the current value and two threshold values that you must set up in the Designer. You can associate a form or URL with a status field. This allows the user to drill-down to a higher level of detail, and examine the data represented by the field.

The following are examples how you might use a status field:

- To provide a Key Performance Indicator (KPI) dashboard.
- To display the importance of a folder when it is opened.

For an example of the status field used in a KPI dashboard, refer to the sample administrative procedure, KPI Dashboards.xep.

To set up a status field you must specify:

- A safety threshold
- A danger threshold
- A value to compare against the thresholds.

 © Metastorm Inc., 2008

The Safety threshold defines the end range of the Safety state, while the Danger threshold defines the beginning of the Danger state. The following diagram illustrates how the Safety and Danger thresholds are used to determine the state of the field:

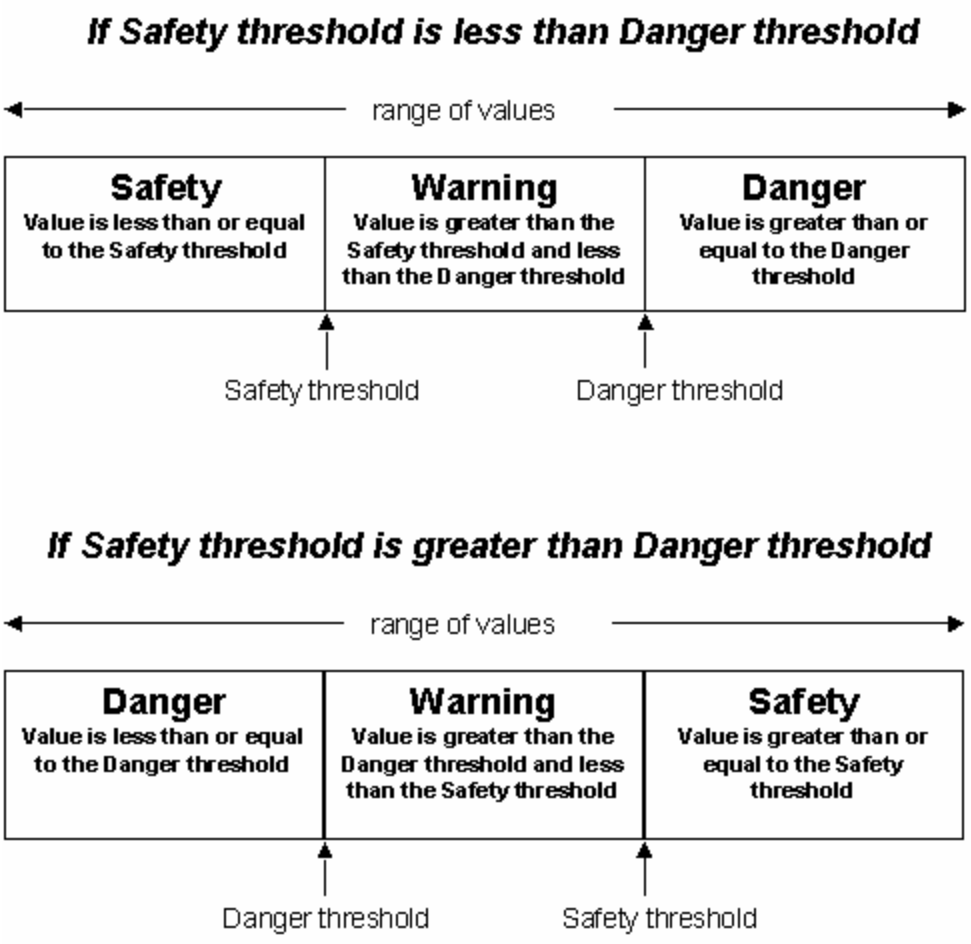

If Safety threshold is less than Danger threshold
range of values
Safety
Value is less than or equal to the Safety threshold
Warning
Value is greater than the Safety threshold and less than the Danger threshold
Danger
Value is greater than or equal to the Danger threshold
Safety threshold
Danger threshold
If Safety threshold is greater than Danger threshold
range of values
Danger
Value is less than or equal to the Danger threshold
Warning
Value is greater than the Danger threshold and less than the Safety threshold
Safety
Value is greater than or equal to the Safety threshold
Danger threshold
Safety threshold

Figure 133: Status field states

If the thresholds are calculated to the same value, a Warning state is not possible. If, in addition, the value is the same as the thresholds' values, the status field will be in the Danger state.

The following examples demonstrate how the state of a status field is determined depending on its thresholds and current value.

Example One

The KPI Dashboards sample administrative procedure includes a Process Performance Dashboard administration form. This displays metrics relating to a process. One of these metrics relates to the number of items with a Priority of 1. A status field is used to determine and display the state of this metric. The administrator sets up the thresholds for this using the Set Process Thresholds administration form.

This example assumes the administrator has set the Safety threshold to 10 and the Danger threshold to 20.

The following table illustrates the state of the **Priority 1 items in progress** metric depending on the current number of Priority 1 items:

Current number of P1 items	State
10	Safety
20	Danger
11	Warning
21	Danger
9	Safety
19	Warning

Table 21: Example values and their states for Priority 1 items in progress

The following diagram illustrates, in pictorial form, into which states these example values fall:

Figure 134: Example values and their states for Priority 1 items in progress

Example Two

In a Sales Lead process, a form may contain a status field used to flag the state of the number of current leads. In this case, if the number of leads was high, the status field should show the Safety state, and if the number of leads had dropped too low, the field should show the Danger state.

Therefore, the Safety threshold would be higher than the Danger threshold.

For this example, assume the Safety threshold is set to 20 and the Danger threshold is set to 10. The following table illustrates the state dependent on the current value:

Current Value	State
10	Danger
20	Safety
11	Warning
21	Safety
9	Danger
19	Warning

Table 22: Example values and their states for a Sales Lead example

 © Metastorm Inc., 2008

The following diagram illustrates which states these example values fall into in pictorial form:

Table 23: Example values and their states for a Sales Lead example

Example Three

In a Sales process, it may be useful to show the status of sales of various products on a form such as the following:

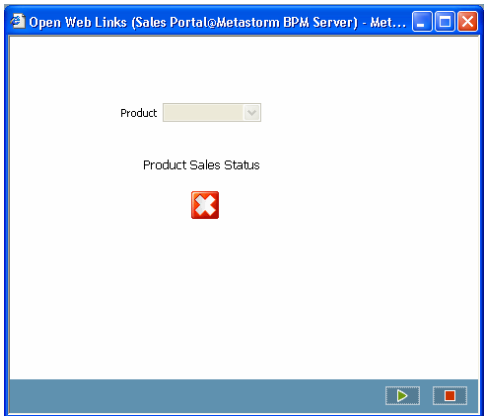


Figure 135: Example Metastorm form showing use of status field

The user selects a product from the dropdown, and the status field indicates the status of the sales for that product.

To allow the same status field to be used for each product, the Safety and Danger thresholds are defined as percentages of the target value. The target value and current value are calculated using values from a database, depending on the chosen product.

For this example, assume that the Safety threshold is 50% of the target value, and the Danger threshold is 10% of the target value. If the target value calculates to 120, and the actual number of sales falls below 10% of 120 (i.e. 12), the status field will be in the Danger state. If the actual number of sales is greater than 50% of 120 (i.e. 60), the status field will be in the Safety state. Otherwise the status field will be in the Warning state.

To add a status field, click on the **Status** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a status field on the form and displays its properties in the Properties editor.

A status field is always read only.

The Properties editor displays five tabs for defining status field properties:

- Status
- Options
- Do this
- Extra
- Notes

For details of the Extra tab, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Status Field's Status *Properties*

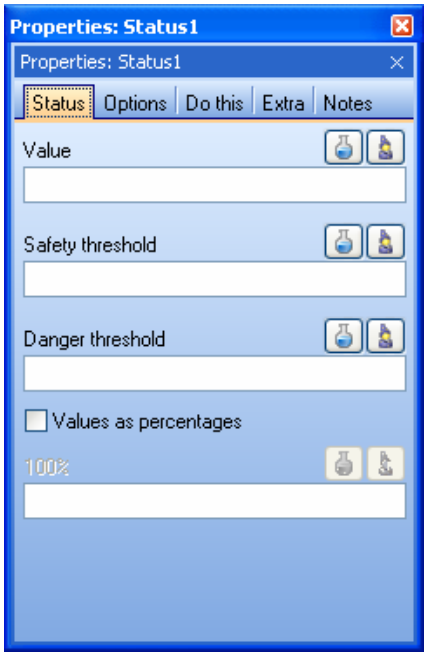


Figure 136: Status Tab

1. Enter the **Value** of the field as a formula.
2. Enter a value for the **Safety threshold** as a formula.
3. Enter a value for the **Danger threshold** as a formula. The Danger threshold can be greater or less than the Safety threshold.
4. You can enter the value and thresholds as real values or percentages. If you choose to enter percentages, check the **Values as percentages** check box.

 © Metastorm Inc., 2008

5. If you choose to enter values as percentages, you must specify the target value in the **100%** property. The threshold values will be treated as percentages of this target value and can be greater than or less than this value. The target value can be calculated using a formula.

These values may be entered as formulas. Remember that scripts can be called via a formula in order to retrieve data from other systems.

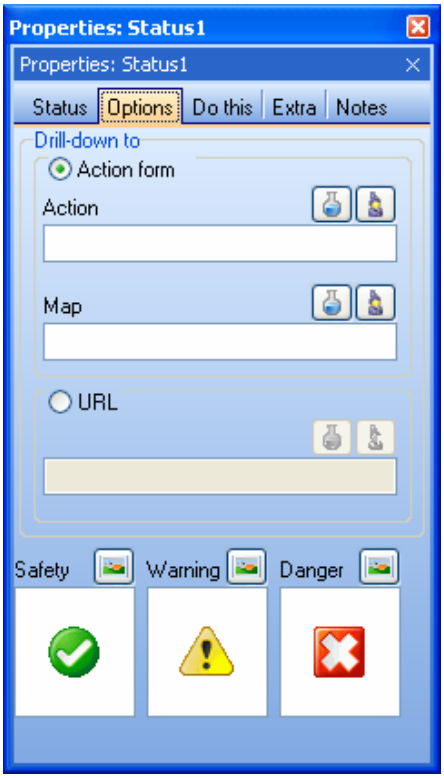


Figure 137: Status / Options Tab

Defining a Status Field's Options *Properties*

Use the *Options* tab to specify drill-down options and custom graphics.

1. To allow the user to **Drill-down to** a form, select the **Action form** radio button and enter the name of a blank form or admin form and the map or admin form group in which it occurs.
2. To allow the user to **Drill-down to** further information via a URL, select the **URL** radio button and enter the URL in the text box.
3. Designer provides default **Safety**, **Warning** and **Danger** pictures for the status field. To replace these pictures, click on the **More pictures** button associated with each picture. To reset a picture to the default, right-click on the picture and select **Reset Image**.

The default pictures are in GIF format. If an alternative format is used, the field may not appear as expected when viewed in the Web Client.

Defining a Status Field's Do this *Properties*

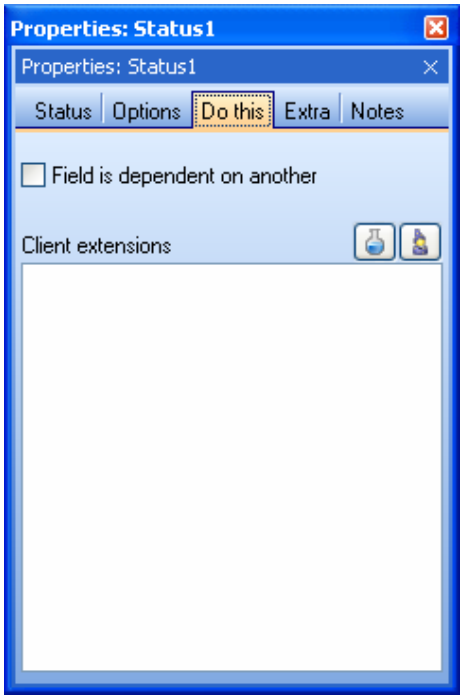


Figure 138: Status / Do this Tab

Use the *Do this* tab to specify dependencies, and processing to be carried out if the contents of the status field change.

1. If the status field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, check the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. Enter any commands to be performed by the system when the status field is entered or exited in the **Client extensions** field. You may create and edit the formula using either of the following tools:

   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.

Defining a Text Field

Use a text field to display or edit a text value or to display a URL in a web browser.

To add a text field, click on the **Text** button on the form toolbar and then click on the location on the form where you want the field to be displayed.

Designer places a text field on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining text field properties:

 © Metastorm Inc., 2008

- Text
- Options
- Do this
- Extra
- Notes

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

Defining a Text Field's Text *Properties*

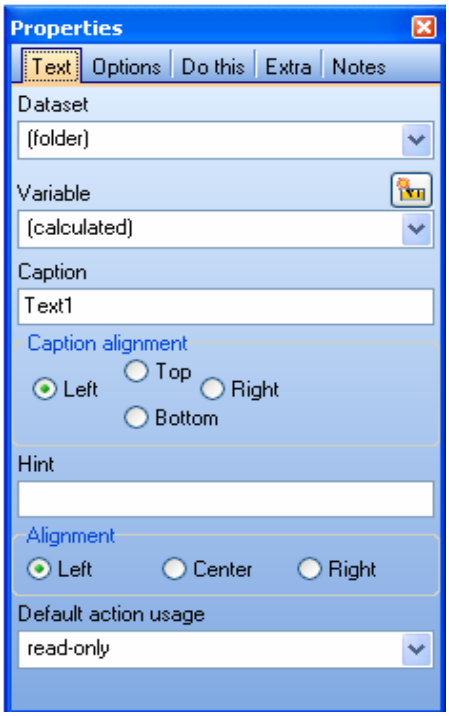


Figure 139: Text Tab

1. The **Dataset** field defines a dataset for the system to use with the text field. The default option is *(folder)*, which indicates that you will not be using a dataset with this field.

   Using a dataset means that any required information necessary for calculating the information displayed in this field will be read directly from the dataset rather than require an additional call to the database. Any datasets previously defined for use with this form will be displayed in the drop-down list. For more information on creating datasets, see the *Defining a Dataset Field* topic in this section.

2. If you wish to change the caption, enter the new caption in the **Caption** field. If you omit this, or have selected *(calculated)* as your variable, the text field will be displayed with a default caption of "Text" plus an auto-incremented number.

3. Text within the text box will be displayed according to the alignment option selected in **Alignment** field.

4. You can include a text string to give users a brief help message when the cursor rests over the field. Type the message you want displayed in the **Hint** field.

5. The **Caption alignment** field contains a radio group listing the following alignment position options:
    - **Left** – Places the caption to the left of the text field.
    - **Right** – Places the caption to the right of the text field.
    - **Top** – Places the caption above the text field.
    - **Bottom** – Places the caption below the text field.

   Select the caption alignment desired.
6. Identify the variable to be used for this field in the **Variable** field. There are a number of options:
    - If the text is to be calculated at run-time, select *calculated* from the list. You will be able to specify how it is calculated on the *Do this* tab.
    - If the text is to be stored in the database, you can select a variable from the drop-down list. Only text or integer variables will appear in the list.
    - You can create a new custom variable by clicking on the **New custom variable button** and specifying the details of the variable. You cannot create a new custom variable if you are using a dataset for this text field.
    - If you are using a dataset, select the column name the system should use to calculate the information for this field from the column names shown in the drop-down list.

   The variable you select on this tab of the Properties editor will determine the fields that appear on the text field's *Do this* tab.

   If you are using a variable other than *(calculated)*, Designer uses the variable name as the default caption for the text field.
7. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, hidden, or **required**—for the Text field. Use the down-arrow associated with this field to select the desired default.

Defining a Text Field's Options Properties

The format of data to be entered into a text box at runtime is defined in the *Options* tab of the text box *Properties*.

To specify the data entry format for the text field:

 © Metastorm Inc., 2008

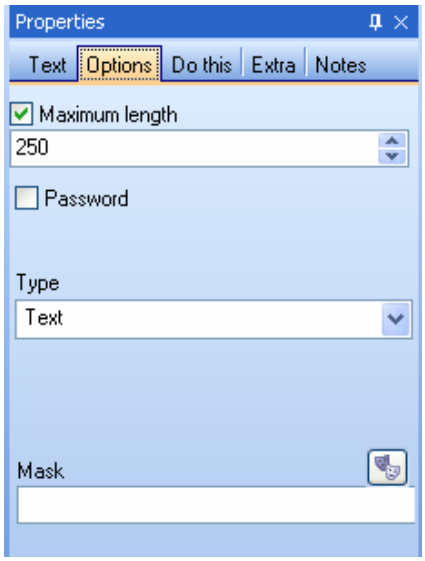


Figure 140: Text / Options Tab

1. Select the text box and select the **Options** tab in the **Properties** editor;
   - The **Properties** editor displays the format options, see *Figure 140*.
2. Set the **Type** field to the default setting**, Text**.
3. To specify the maximum number of characters that can be entered into the text field, you must check the **Maximum length** checkbox. In the field below, select a value between 1 and 250.
4. If you check the **Password** check box, Designer is instructed to store all user input in the database in encrypted form. A password will never be provided by the server; it must be provided by the user.

   *The **Password** option* does not provide any authentication *of user input. Password authentication would have to be requested separately from the server, such as through a formula in the* When changed *property on the text field's* Do this *tab that required the server to compare the field input with a database value.*

5. The **Mask** field defines the entry format of the text that can be entered on a client machine. Text that does not conform to the **Mask** will not be accepted.

*The **Mask** option is not available if the **Password** field is checked.*

Masks can be typed directly into the **Mask** field or they can be selected from a list of options, e.g. Time of day. To set a predefined map for a text entry:

a) In the **Options** tab click on the Mask Icon ;
   - A **Mask Picker** dialog box is displayed, see *Figure 141* below;
   - Under the **Mask Name** is a list of preset masks.

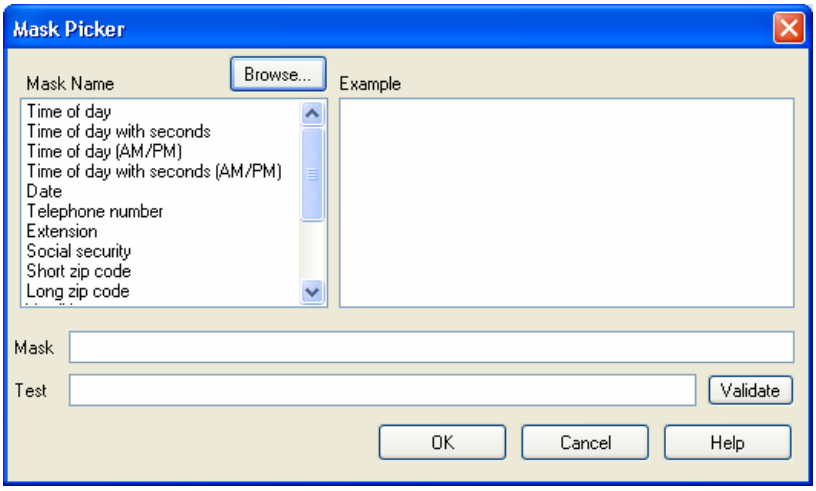


Figure 141: Mask Picker Dialog

b) Select the required Mask from the list;

- The **Example** text box displays samples of the selected Mask; and
- The Mask is defined as a regular expression in the **Mask** text box.

c) Test the selected Mask by entering data into the **Test** text box and click the **Validate** button;

- If the data is in the correct format, an acceptance message is displayed:

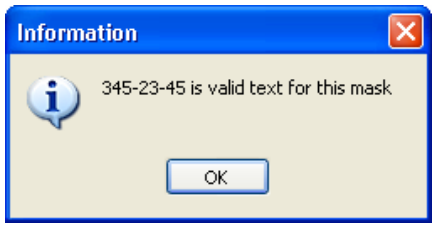


Figure 142: Data acceptance using Mask Picker

- If the data is incorrect, a rejection message is displayed:

Figure 143: Data rejection using Mask Picker

d) When satisfied that the Mask is correct click **OK**;

- The selected data mask is displayed in the **Mask** entry box of the **Properties** editor; and
- Any data entered into the **Test** field is discarded.

 © Metastorm Inc., 2008

e) If there are no suitable formats in the **Mask Name** list, select:

- **Browse,** using the **Browse** button, to a folder with a suitable mask collection (file type of *.dxm') file;

For details of the DXM file format, refer to Appendix E – Masks Format.

If the text field is an optional *field on the form, the user may leave the field blank and still have the field pass validation, even though a text mask property has been defined. If the mask set in the Properties editor represents required input, the field must be made a* required *field on the form.*

For users who want to develop their own Regular Expressions for text masks, refer to http://www.regular-expressions.info/.

6. In the Properties editor Options tab the field type is displayed in the **Type** field. A drop-down list can be used to change the type of field. To do so select the **Type** dropdown List;
   - The dropdown list displays options for Drop-Down, Radio Group, Date/time, and number.
7. Select the type required;
   - The text box on the form changes to the selected type and the Options editor displays the input options.

If you change the field type after you have defined the field's properties, be sure to review the properties for the new field type.

Defining a Text Field's Do this *Properties*

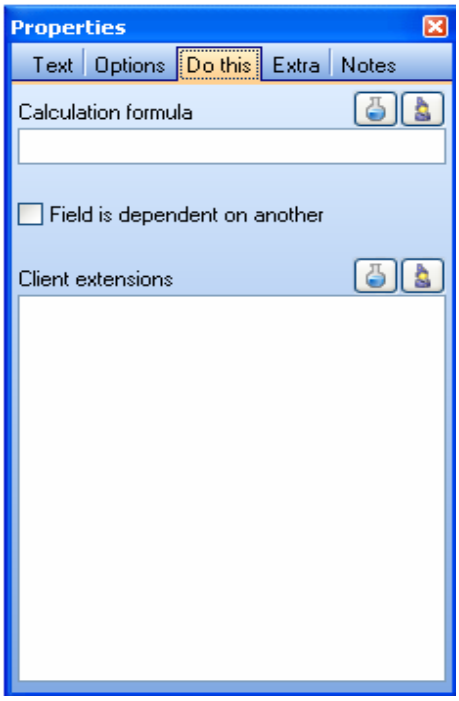


Figure 144: Text / Do this Tab

Use the *Do this* tab to specify the calculation formula, dependencies, and processing to be carried out if the contents of the text field change.

1. If the text field may be updated as a result of a change to another data field or grid that has dependants, or as a result of a button being pressed, click on the **Field is dependent on another** check box.

   If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

2. The **Calculation formula** field allows you to specify how the contents of the text field are calculated. This field is visible only if you selected *(calculated)* as the field's variable. You may create and edit the formula using either of the following tools:

   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.

3. If a custom variable has been assigned to the text field, the **Do this** tab will display a **When Changed** option instead of a calculation formula.

   In the **When changed** field, enter any command the system is to perform when the contents of this list field are changed. For the commands in the **When changed** field to be performed when the field is updated, the **Field has dependants of its own** checkbox must be checked.

4. If you selected a system variable or created a new custom variable for this text field, the **Field has dependants of its own** check box will be displayed on this tab of the Properties editor. This field instructs the system to notify immediately the Process Engine of any change to this text field. This is necessary if any other fields on the form are dependent upon the content of the text field.

   This box must be checked *if any other fields are dependent upon values in this field.*

5. Enter any commands to be performed by the system when the text field is entered or exited in the **Client extensions** field. You may create and edit the formula using either of the following tools:

   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.

Using a Text field to add a link to the form

There are two ways in which you can add a URL link to a form using a text field:

 © Metastorm Inc., 2008

Figure 145: URL in Do this tab

1. Using the calculation formula.
    a. Add a text field to the form.
    b. Set the **Caption** field on the *Text* tab. The link that appears on the form will have this caption.
    c. Ensure the **Variable** field on the *Text* tab is set to 'Calculated'.
    d. On the *Do This* tab, set the **Client Extensions** to 'URL'. Enter the URL in the **Calculation Formula** field.

Using the calculation formula is only available in the Web Client and the Windows Client using web forms.

2. Assigning the URL to the field's custom variable.
    c. Add a text field to the form.
    d. Set the **Caption** field on the *Text* tab. The link that appears on the form will have this caption.
    e. Associate a custom variable with the field using the **Variable** field on the *Text* tab.
    f. On the *Do This* tab, set the **Client Extensions** to 'URL'.
    g. At an appropriate point in the map, assign the custom variable with a URL. For example, add the following formula to an action's **When action started** field: `%URLTextVariable:="http://www.metastorm.co.uk".`

These two methods will produce a link on the form when the client accesses it.

Figure 146: Text field links on a Metastorm form

It is not possible to change the URL at runtime.

Defining a Grid

You can place a table (referred to as a *grid* within Designer to differentiate it from a database table) on any form. A grid maps multiple rows and columns (fields) from one or more database tables. You can place any number of grids on a form. Grids will overlay any background (image, rule, or label) elements.

If a grid is published against a case insensitive SQL Server database (or an Oracle database), it can be run only against a case insensitive database (or an Oracle database).

For example, if the EUSER column is used in a grid, and the procedure is published against a case insensitive database, the procedure may only be run against a case insensitive database.

If the procedure is subsequently to be run against a case sensitive SQL Server database, the column (and any other columns of the grid) must be redefined using a case sensitive database.

i.e. the EUSER column should be redefined as eUser.

This applies to anywhere that table names and column names are used.

If you load a procedure containing a grid, the properties of the grid may be loaded incorrectly, if any of the columns contain subqueries in the SQL.

To place a grid on a form, click on the **Grid** *button* on the Form toolbar and then click on the location on the form where you want the grid to be placed.

Grid Types

There are four types of grid:

- '**Field has dependants**' grid
  A refill occurs each time the client user double-clicks (in Windows Clients) or single-clicks (in the Web Client) on a 'Field has dependants' grid.
- '**Edit**' grid

 © Metastorm Inc., 2008

The client user may add or remove rows and update the data and in an 'Edit' grid.

- '**Folder**' grid

  The Folder on Folder feature may be used with a 'Folder' grid.

Refer to the section on Using a Grid to set up Folder-on-Folder for further details.

- '**None of the above**' grid

  This is a basic grid. No refills will occur when the user clicks on a 'None of the above' grid. If formatting is required, the appropriate column type must be selected.

Refer to the section on Defining a Grid's Columns Properties for information on selecting column types.

To choose the type of grid, click on the relevant **Grid Type** radio button on the Grid's *Do this* tab.

Designer places a grid on the form and displays the corresponding Properties editor.

The Properties editor displays five tabs for defining grid properties:

- Grid.
- Columns.
- Do this.
- Extra.
- Notes.

For details of the Extra and Notes tabs, refer to section 6.4.3 "Defining Element-independent Properties" on page 139.

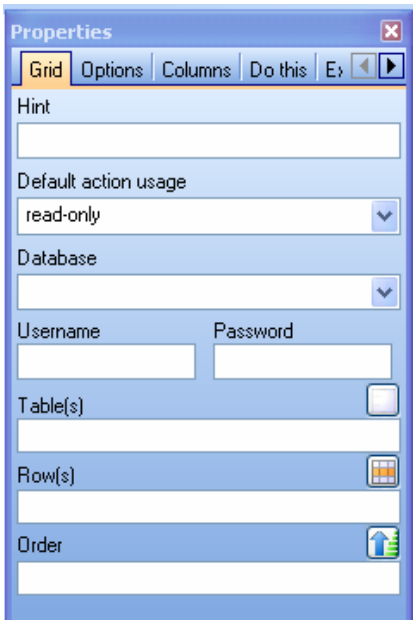


Figure 147: Grid Tab

Defining a Grid's Grid Properties

1. If you wish to retrieve data from an external database, enter the name of the ODBC Data source in the **Database** field. This name must have been set up on any server to which the procedure will be published, using the ODBC Data Source Administrator. If you want to be able to browse this database (for tables and fields) from within Designer, then the Data Source Name (DSN) must also have been set up on the computer on which Designer is running.

   If you wish to retrieve data from the local Metastorm database, you should not specify a DSN.

2. If this database needs a **User name** and **Password** to enable connection, enter these in the appropriate fields.
3. Enter the name of the database table or tables (from the data source identified in step 1) containing the data to be displayed in the **Table(s)** field. To see the Pick tables dialog with a list of the available tables, click on the **Pick table(s)** button.

   Database table names that include spaces are not supported.

   The **Pick Table(s)** dialog box identifies the database you are looking at and displays a list of the available tables in the scrolling box on the left. Select the tables you wish to draw information from and move them into the *selected* field in the center. You may do this in a number of ways:

   - **Selecting Tables**:

     Select a table by clicking on the individual table name.

     Select multiple tables by clicking on the first desired table and then **Shift + Click** to select additional tables in sequence, or **Ctrl + Click** to select additional tables out of sequence.

   - **Moving Selected Tables**:

     Click and drag the selected tables into the "selected" field.

     Use the > button to move the selected tables into the "selected" field.

     Use the >> button to select all tables in the list and move them into the *selected* field.

4. In the **Row(s)** field enter the body of an SQL WHERE clause to specify which row(s) are to be included from the table. For example, you might enter

   ```
   eUsername = '%user.name'
   ```

   to pick only rows relevant to the user of a form. Click on the **Pick Rows** button to open the Condition builder dialog to help you build the SQL statement.

   If you do not specify the row(s), Metastorm BPM displays all rows from the selected table.

5. You may use the **Order** field to determine the order of the grid columns. Clicking on the Pick fields button attached to this field will open the Pick fields dialog box.

 © Metastorm Inc., 2008

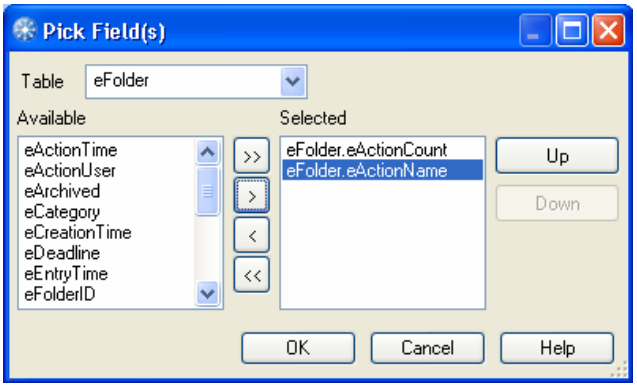


Figure 148: Pick Fields Dialog Box

The **Pick Fields** dialog box identifies the table specified above and displays a list of the available fields in the scrolling box on the left. Select the fields you wish to use to sort the rows in the grid and move them into the *selected* field on the right.

To specify the order in which fields should be used to order the rows in the grid, select a field name in the *selected* list and press the **Up** or **Down** button to reposition it. For example, if you wish to sort name within department, place department higher in the list than name.

6. The **Default Action Usage** field allows you to specify a default—such as **read-only**, **optional**, **hidden**, or **required**—for the Grid field. Use the down-arrow associated with this field to select the desired default.

*The **Default Action Usage** for an editable grid cannot be **required**.*

In the Web client, a read-only grid can be viewed page by page. The number of rows on a page depends on the user's Browser Settings. Refer to the 'Using Metastorm BPM with Internet Explorer' guide for further details.

Please note when using grid-specific client-side script functions with a paged grid, the RowCount refers to the number of rows in the current page, not the number of rows in the grid as a whole. In the same way, the Row index refers to the row's index in the current page, not in the grid as a whole.

7. You can include a text string to give users a brief help message when the cursor rests on the grid. Type the message you want displayed in the **Hint** property.

Defining a Grid's Options Properties

The *Options* tab in the Grid's *Properties editor* provides the facility to define the delimiter to be used in the drop-down list of *Columns'* names. The selected delimiter is applied to all columns in the grid.

Figure 149 - Grid delimiter selection

To define the delimiter used in a Grid:

1. Select the **Options** tab;
    - The Options property editor is displayed.
2. Enter the delimiter by one of three methods:
    - Type directly into the Delimiter field;
    - Use the Integration Wizard; or
    - Use the Formula Editor.

For further details on how to enter a delimiter see page *174*.

Defining a Grid's Columns Properties

A grid is set up initially with one column with the default name, **Column 1**. To specify the columns to be displayed, select the *Columns* tab.

 © Metastorm Inc., 2008

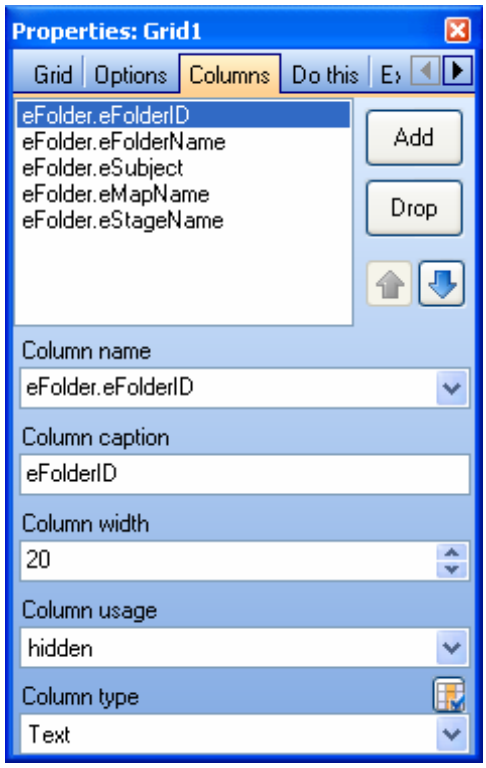


Figure 150: Grid / Columns Tab

1. Add further columns by clicking on the **Add** button. If you want to remove a column select it and click on the **Drop** button.
2. When you select a *column* from the list, Designer places its properties into the remaining boxes on this tab. The **Column name** drop-down list contains the name of the column currently selected.

Do not accept the default column name created when a new column is added. The column name must match a valid column in the selected table or the form with the grid will not load. Selecting the column name from the drop-down list ensures that the column name is valid.

The Oracle syntax for concatenating columns using the pipe operator is not supported and will result in an 'Unable to load xml into DOM' error from the Engine. The work around is to create a column definition using the CONCAT command For example: CONCAT(MYTBL.COL1,MYTBL.COL2).

3. By default the caption is the same as the column name. You can specify a different caption to be displayed for each column of the grid in the **Column caption** property. The caption is displayed in the top (fixed) row of the column, left justified.

4. Use the **Column width** option to specify the width of the column in pixels. A column will not display narrower than its caption.

5. Use the **Column usage** option to specify whether the column will be **required**, **optional**, **read-only**, or **hidden**.

6. Use the **Column type** dropdown to select the data type of the column.

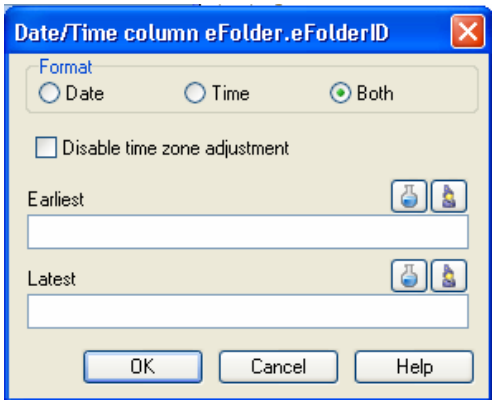
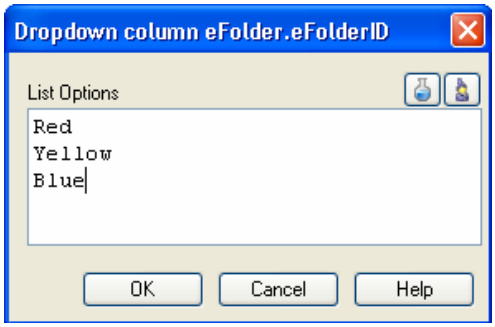
Column types should be mapped only to the database field types shown in the following table:

Grid Column Type	*Database Field Type*
Date/Time	Date/Time
Text	Text, Memo, Date/Time, Integer, Real, Currency
Number	Integer, Real, Currency
Dropdown	Text
Currency	Integer, Real, Currency

*Table **24**: Grid column type properties*

When generated by a formula, the items in a grid's drop-down column can be in the form of a set of single values or label/value pairs. When label/value pairs are used, the labels are shown in the drop-down and the values are passed to the database. You cannot specify a static list containing a label/value pair.

7. Selecting the relevant column type allows the grid cells to be formatted according to their type. To set the formatting options for the column, click on the **Column Type Properties** button. The formatting options shown in the following table are available:

Column Type	*Column Type Properties dialog*	*Column Type properties note*
Date Time		f you select the **Date** or **Both** format, the **Earliest** and **Latest** options become available. If you select the **Time** or **Both** format, the **Disable time zone adjustment** checkbox is available. If you leave this unchecked, the values in this column will be displayed in the client's local time.
Dropdown		To populate the cells in a dropdown column, enter the values into the **List Options** box, or use the Integration wizard or Formula Editor to populate the list at runtime. For further information, refer to the information on defining drop-down fields.

 © Metastorm Inc., 2008

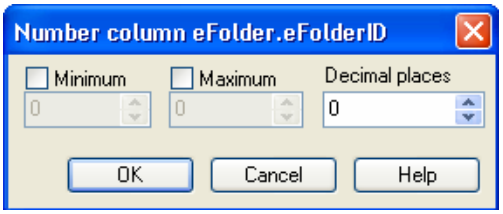
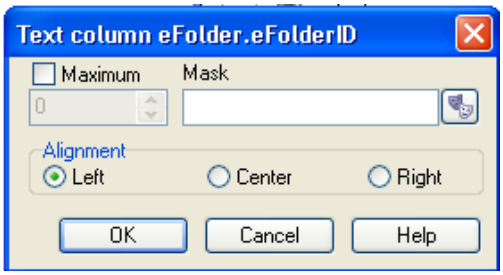
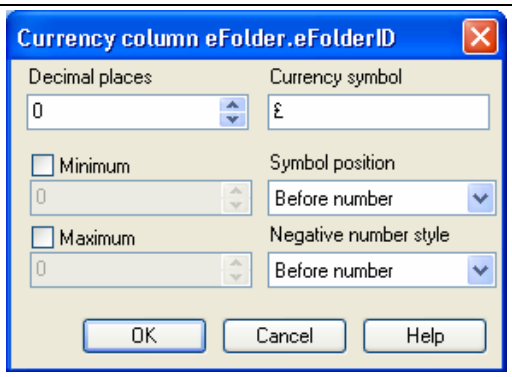
Column Type	*Column Type Properties dialog*	*Column Type properties note*
Number		Check the **Minimum** or **Maximum** checkbox to select the minimum or maximum value allowed for the field, if required. Select the number of **Decimal places**. The values entered into this column will be formatted to the specified number of decimal places.
Text		Check the **Maximum** checkbox and enter the maximum number of characters allowed in the cells of this column. The maximum value is in the range 1 to 250. Select the mask icon to enter a text **Mask**. *Refer to section on Defining a Text Field's Options Properties for information on Masks.* Select **Left**, **Center** or **Right** to set the alignment of the text in the cells.
Currency		Refer to the section on *Defining a Currency Field's Currency Properties* for information on the currency column property options.

Table 25: Grid column type properties

8. Repeat these steps for each column in the grid.

Defining a Grid's Do This Properties

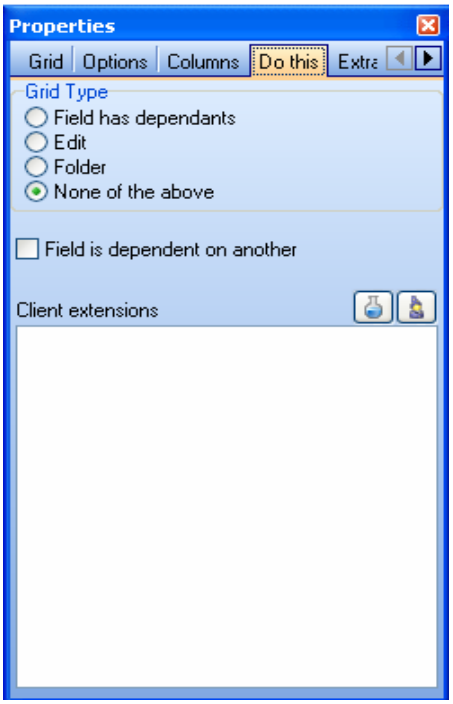


Figure 151: Grid / Do this Tab

Select the grid type by selecting it from the **Grid Type** radio group.

The options that are available on the *Do This* properties tab depend on the type of grid you are defining.

1. Irrespective of the grid type, the **Client Extensions** option is available and allows you to enter any commands to be performed by the system when the grid is entered or exited or when a cell is exited. You may create and edit the formula using either the Integration wizard or the Formula Editor.
2. If you are defining a *Field has dependants*, *Folder* or *None of the above* grid, you may specify whether the **Field is dependent on another**. If this checkbox is checked, the grid may be updated as a result of a change to another data field, or as a result of a button being pressed.

If the field is dependent on another, the field it is dependent on must have the Field has dependants of its own *option selected on its* Do this *tab.*

3. If you are defining an *Edit* grid, the options to allow users to **Insert** and **Delete** rows in the grid are available.
4. If you are defining a *Field has dependants* grid, you can specify the processing in the **When user selects row** property. You may create and edit the formula using either the Integration wizard or the Formula Editor.

 © Metastorm Inc., 2008

Using a Grid to set up Folder-on-Folder

Folder-on-folder is a feature that allows you to open any existing folder from within an already open folder. It is most commonly used to allow you to view the status of child folders (folders generated by sub-procedures of the current procedure). You may implement folder-on-folder by placing a grid on a form in your procedure and setting its properties to values Designer recognizes as specific to a folder-on-folder implementation.

The New Hire sample procedure (*Designer\Sample Procedures\New Hire.xep*) provides an illustration of this technique on the form *Sub process data* that may be found under the *New Hire/Forms* item in the Procedure Explorer.

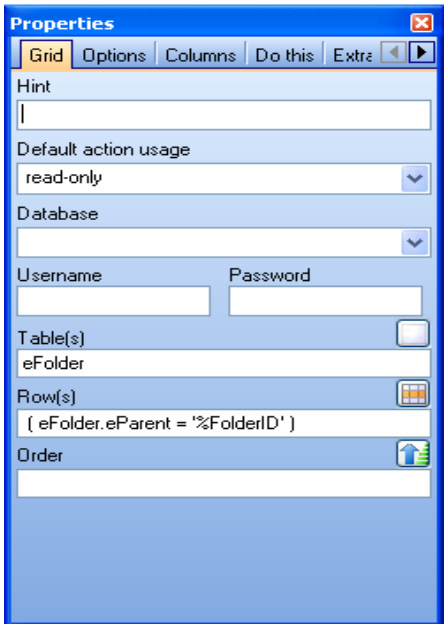


Figure 152: New Hire Sample Procedure, "Children" form, "Grid" Properties editor—Grid Tab

To establish folder-on-folder:

1. Place a Grid on your form.
2. Open the Properties editor.
3. On the **Do this** tab select the *Folder* Grid Type.
4. Complete the fields on the **Grid** tab. The following fields require special handling:

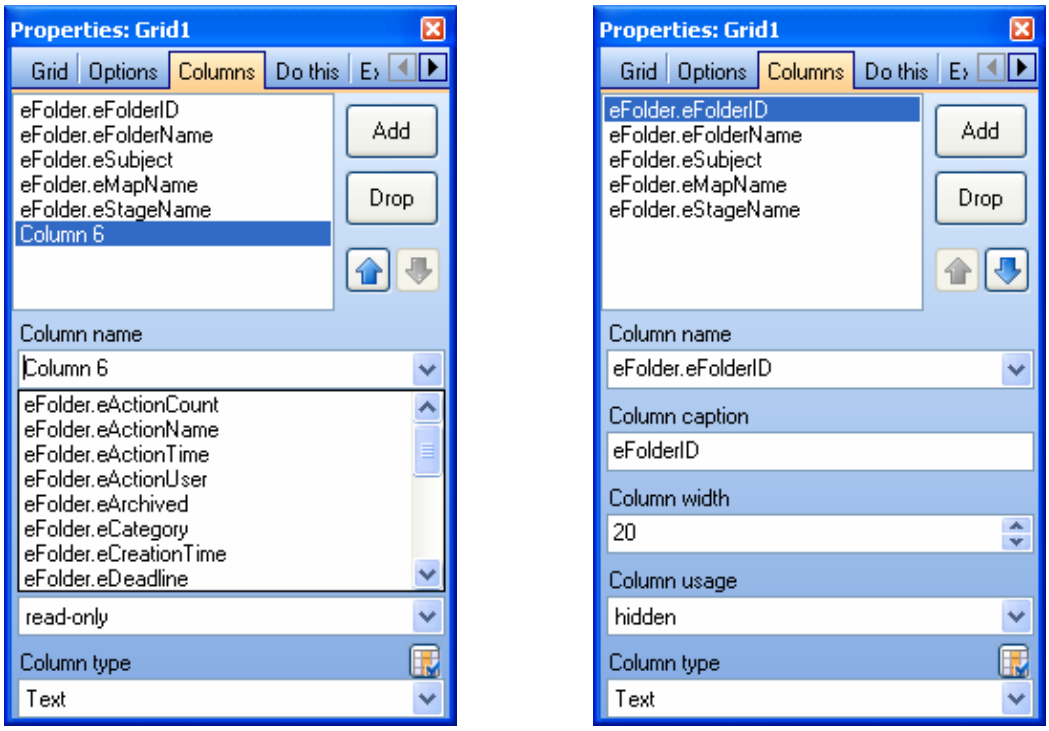
    - The **Table(s)** field – you may select any table that contains an *eFolderID* field containing folder information. In the New Hire example, the Table name is "eFolder."
    - The **Row(s)** field – if you wish to specify which row(s) are to be included from the table, you would enter the appropriate SQL statement here. For example, as shown in Figure , the New Hire sample procedure, the formula is:

```
(eFolder.eParent='%FolderID')
```

This formula instructs the form to search the table (eFolder) for the rows whose parent (eParent) is the current folder (%FolderID).

5. Select the Properties editor, **Columns** tab.

6. Complete the fields on this tab. For Column 1, the following fields require special handling:



Column Name *Column Caption Field*

Figure 153: New Hire Sample Procedure, "Children" form, "Grid" Properties editor – Column Tab

- The **Column name** drop-down list – select **Column 1** from the columns list. Scroll through the available **Column names** until you find the desired column name. The "Column 1" designation will be replaced with the column name you selected.

For Column 1, any column name may be selected from the column name drop-down list, as long as the column contains valid folder IDs. This is most easily identified in columns named [tablename].eFolderID*, as illustrated in the New Hire sample shown above. A thorough knowledge of your table structure is recommended before selecting a column name that does not identify itself as containing valid folder IDs.*

- The **Column Caption** field – when it is the first column in a grid, Designer does not display the *eFolderID* column by default. However, the **Column Caption** for Column 1 must read *eFolderID* for Folder-on-Folder to work.

7. Select any additional columns you wish to have displayed. No special handling is required for defining the additional columns. In the New Hire sample, the columns *eFolderName* (name of the folder), *eSubject*, *eMapName* (name of the submap), and *eStageName* (name of the stage the folder is currently at) were selected.

 © Metastorm Inc., 2008

Avoiding Common Editable Grid Errors

This section outlines issues that process designers should be aware of when working with grids.

- **Avoid Primary Key Violations**

  A primary key violation will occur if a user tries to submit a form whose editable grid has two rows with the same value for the table's primary key.

  For example, a table, TableOne, has two columns; FieldOne and FieldTwo. FieldOne is the primary key of the table.

  Attempting to submit the following editable grid will cause a primary key violation.

FieldOne	***FieldTwo***
A	Hello
B	Metastorm
B	**World**

Table 26: Editable Grid

  The user must re-edit the grid so that the values for the primary key field are unique.

- **Avoid Concurrency Violations**

  Concurrency errors occur if the data in the database is altered while the grid is being edited.

  When an editable grid is generated, a snapshot of the values in the database is made. While working, any insertions, deletions or updates are made to the snapshot. When an editable grid is submitted, the values in the snapshot are compared with the current values in the database (for every delete or update). If they differ, the update will fail with a concurrency error.

  This can be avoided in the following ways:

  - Do not include two or more editable grids that refer to the same data on the same form.
  - Ensure that data displayed in an editable grid is not highly volatile.
  - Ensure the editable grid data is displayed based on the folder's folder-id.

- **Avoid the Cursor Error, 'Too many rows affected'**

  This error may occur when an editable grid is based on a table that has no primary key defined. For example, a table, TableTwo, has two fields; FieldOne and FieldTwo. A primary key has not been defined.

  The user opens a form and is presented with the following grid:

FieldOne	***FieldTwo***
A	Hello
A	World
A	World

Table 27: Grid on a Form

The user modifies the grid to:

FieldOne	*FieldTwo*
A	Hello
A	Metastorm
A	World

Table 28: Modified Grid

The system has no way of determining which row has been modified and a "Too many rows affected" error occurs.

To avoid this error, ensure that the tables referred to in an editable grid have primary keys defined. (If inserts are not required, the primary key need not be displayed).

- **Avoid errors when loading forms containing grids**

If the Rows property of a grid uses a variable, and this variable has no value when the form is loading, an error occurs.

Avoid this error by ensuring that the Rows property has a valid default value when the grid is initially loaded on the form.

6.5 External Forms

You may design forms using the Designer, or you may import them from any HTM/HTML or PDF form designer. External forms are supported only in the web client.

6.5.1 Defining External Forms Options

In order to use an external form, Designer requests the following information about the external form and its associated interface:

- The external form file type (extension).
- A one-line description of the file type (optional).
- The path and filename of the external forms designer application (optional).
- The path and filename of the external forms interface DLL.

The file type, description, and DLL information for HTM/HTML and PDF form types is contained in a predefined list. However, you may wish to include information about your external forms designer application for convenience in editing your form while creating your procedure. Information about external forms may be reviewed, and modified through the Designer *External Forms* dialog.

External forms definitions are maintained in a master list used for all procedures created by the Designer. If you add, delete, or modify the external forms definitions in the External Forms Options list, this list will affect other procedures developed or modified under the same installation of Metastorm BPM.

To Add, Modify, or Review External Forms Options

1. Click on **View | Options** from the menu.
    - The **Options** dialog appears.

 © Metastorm Inc., 2008

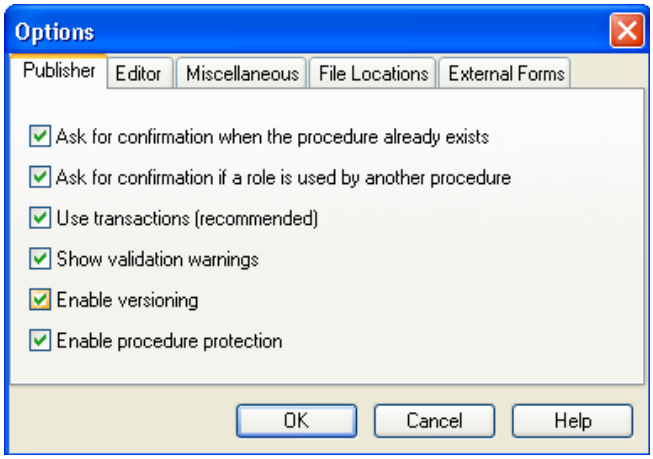


Figure 154: Options Dialog

2. Select the External Forms tab from the Options Dialog
    - Designer displays the Options dialog showing the forms types:

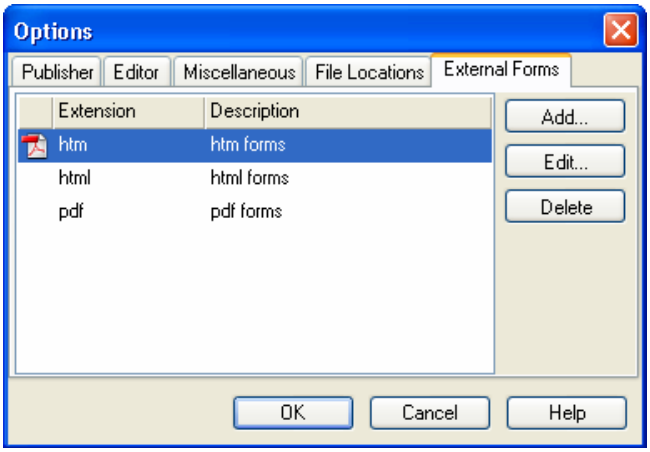


Figure 155: External Forms Dialog

If any External Form file types have been previously defined, they will be displayed in the dialog screen.

3. To review the properties of one of the predefined file types, select the file type from the list and then click on the **Edit** button. To add a new file type to the list, click on the **Add** button.
    - Designer displays the **External Form** dialog box. If you have selected one of the predefined file types, most of the fields will already have information displayed. If you are adding a new file type, all of the fields will be blank.

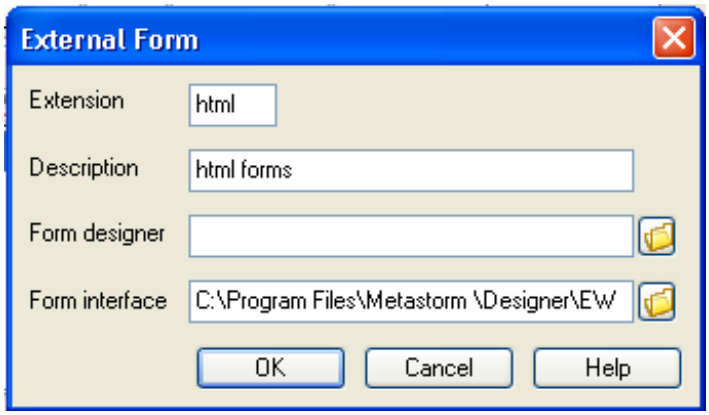


Figure 156: External Form Properties Dialog

4. In the **Extension** field, type the file extension for the new file type.
5. In the **Description** field, type a brief description of the file type. This description will be displayed in the main External Forms Options list.
6. In the **Form designer** field, enter the path and filename of the application used to create and edit the new file type. Selecting the **Browse** button on this line will open a standard **File Open** dialog allowing you to browse for the application file.
7. In the **Form interface** field, enter the path and filename of the external forms' interface DLL.

If you have installed Metastorm BPM using the standard default file locations, the DLL is found in C:\Program Files\Metastorm \Designer

The DLL for HTM/HTML forms is EWHTML.DLL

The DLL for PDF forms is EWPDF.DLL

8. Before any external forms may be used within a procedure, the **Extension** and **Form interface** fields *must* be completed. The **Form designer** field is optional, and provided for your convenience in editing your form. When you have completed your entries, click on the **OK** button to save your entries, or **Cancel** to clear the form and return to the External Forms Options dialog.
9. To remove a file type from the main **External Forms** options list, select the file type and click on the **Delete** button.

There is no Undo option for this action. If you inadvertently remove the wrong file type from the list, you will have to use the Add process to replace it.

6.5.2 Defining External Form Properties

In order to use an external form in a procedure, you must define the filename and other form properties.

 © Metastorm Inc., 2008

To create a form, click on the **New Form** button in the General toolbar, or from the Component menu select the **New Form** option.

Designer adds a form icon to the Procedure Explorer. Designer also displays the Properties editor and the Form toolbar.

The Properties editor displays the following tabs for defining form properties:

- Form.
- Format.
- Do this.
- Notes.

Defining an External Form's Form *Properties*

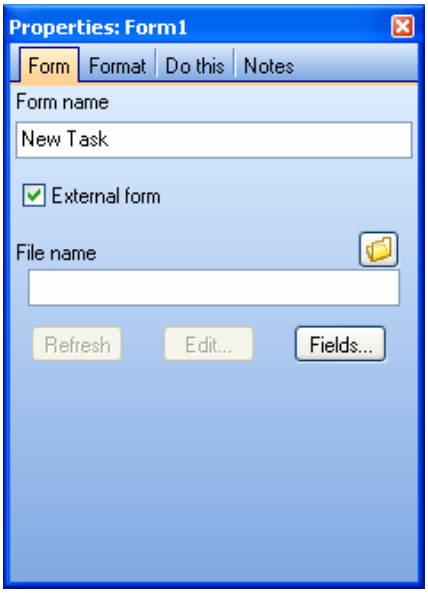


Figure 157: External Form / Form Tab

Forms have several predefined properties which may be reviewed and changed through the Properties editor; some of these properties also apply to external forms.

General properties for a form are defined on the **Form** tab of the Properties editor.

On the Form tab of the Properties editor, enter a short, meaningful name for the form in the **Form name** field. This name is displayed when the user opens a folder that includes this form. This field may not be left blank, and the form name must be unique for each form in a single procedure. The following characters are not supported in the **Form Name** property:

- Pipe (|).
- Period (.).
- Dollar Sign ($).
- Equal Sign (=).

- Back Slash (\).
- Quotation Mark (“).
- Forward Slash (/).
- Percent (%).
- Comma (,).
- Plus (+).
- Colon (:).
- Semi-colon (;).
- Open bracket (<).
- Close bracket (>).

A number of words have special meanings in the databases that Metastorm BPM supports. You should not use any of these reserved words when naming an external form. See the Reserved Words.txt file for a list of reserved words. This file is located in the Metastorm \Designer folder.

1. To use an external form, check the **External form** check box.
    - The Properties editor will change to display the property fields exclusive to an external form.
2. In the **File name** field, enter the path and filename of the form you wish to use. Selecting the **Browse** button on this line will open a standard **File Open** dialog allowing you to browse for the file.
    - When you publish your procedure, Designer will store the file information in the database and make the form available to Metastorm users.

Any time you make changes or edit the form file, you will have to use the Refresh button on this tab of the Properties editor to update the file information stored on the database the next time the procedure is published.

3. To create variables for fields in your external form, click on the **Fields** button.

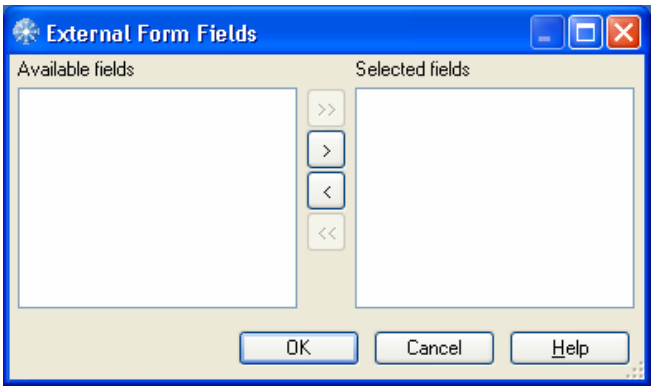


Figure 158: External Form Fields

Designer will display an External Forms Fields dialog box as shown in *Figure 158*.

This dialog provides a list of available fields from the external form, and allows you to select the fields for which you would like to create custom variables.

 © Metastorm Inc., 2008

4. To make changes to your external form, click on the **Edit** button.

   Designer opens the external forms designer application defined for this file type (refer to the instructions for *Defining External Forms Options* on the previous pages). You may edit your external form and save the file according to the conventions of the application.

5. When you have finished making changes to your external form, return to the Form tab of the Properties editor for this form and select the **Refresh** button. To update the file stored in the database, republish the procedure.

Defining an External Form's Format *Properties*

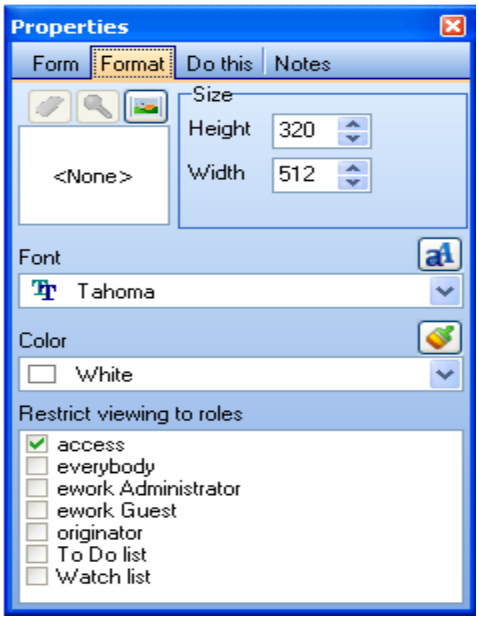


Figure 159: External Form's Format Tab

When you are viewing an external form through the Web Client or Windows Client using web forms, it is displayed to the end-user through the external form's application software. (For example, the system would open a Web browser to view an HTML form.) For users who do not have the appropriate software on their system to view the form in its original state, Designer creates a *shadow form*. A shadow form retains all of the field definitions of the original form, but its layout comes from the properties defined through the **Format** tab of the Properties editor.

1. To define a background image for the shadow form, click on the **Image** button below the **Format** tab. Designer will display a standard **File Open** dialog, allowing you to browse for the image you wish to use.
2. To delete a background image from the shadow form, click on the **Clear** button.
3. To select a background color for the shadow form, scroll through the available drop-down list, or click on the **Colors** button. Designer will display a color palette from which you may select a background color.
4. To select the default font for the shadow form, scroll through the available drop-down list, or click on the **Fonts** button.

5. You cannot change the name or the caption of a field on a shadow form as these are used to map to the fields in the external form.

Formatting defined for a shadow form will not be used *when the form is displayed in its original state. This also applies to any repositioning of fields on the form that you may do through the Form Editor.*

6. In the **Restrict viewing to roles** field, select those roles that should be allowed access to the form. This property definition applies to both the external form and the shadow form.

Defining an External Form's Do This *Properties*

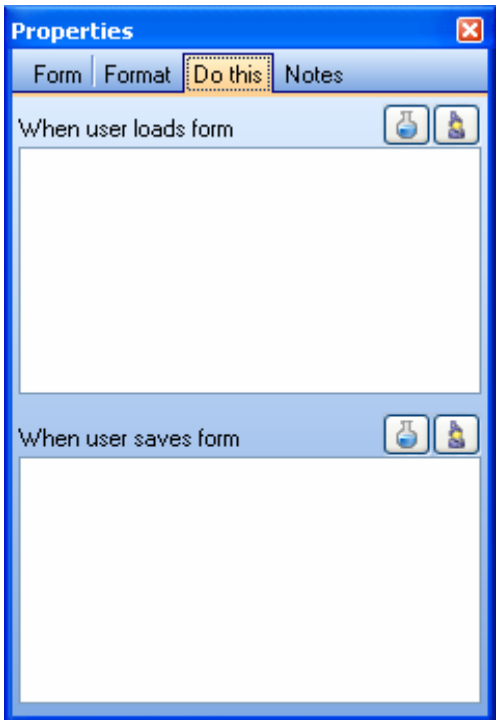


Figure 160: External Form's Do This Tab

Actions that should be performed when the form is opened or closed are defined on the **Do this** tab of the Properties editor.

1. In the **When user loads form** field, enter any commands the system should perform when the form is opened, whether as a tab in a folder or as an action form. You may create and edit the formula using either of the following tools:
   - Click on the **Integration wizard** button to open the Integration wizard.
   - Click on the **Formula editor** button to open the Formula editor dialog.

If the form is being used as part of an action, the When user loads form *command will be performed after the When* action started *command.*

You should not use the When user loads form *property to set field defaults for a blank form. This should be done using the* When action starts *property.*

 © Metastorm Inc., 2008

2. In the **When user saves form** field, enter any commands the system should perform when a user commits (saves) an action that uses this form, whether as a tab in a folder or as an action form. You may create and edit the formula using either of the following tools:

    - Click on the **Integration wizard** button to open the Integration wizard.
    - Click on the **Formula editor** button to open the Formula editor dialog.

If the form is being used as part of an action, commands will be performed in the following order: (1) When action started *(2)* When user loads form, *(3)* When user saves form, *(4)* When action completed. *If the commands fail, the form will not be closed and the action will not be saved. The user may choose to change the data and try again or cancel the action.*

3. In the **Client extensions** field, enter any commands the system must perform when the form is loaded (whether as a tab in a folder or as an action form), submitted or cancelled. You may create and edit the formula using either of the following tools:

    - Click on the **Integration wizard** button to open the Integration wizard.
    - Click on the **Formula editor** button to open the Formula editor dialog.

Defining an External Form's Notes *Properties*

The **Notes** tab of the Property editor displays a memo field which the process designer may use to make useful notes, but which will not be seen by end-users.

To use this field, place your cursor in the field and type your comments.

6.5.3 External Forms Issues

The process designer should be aware of the following issues when designing procedures using external forms.

Euro symbols in PDF forms

Due to differences in the character sets supported by Windows and Adobe Acrobat®, the use of the Euro symbol in fields on PDF forms is not supported. While the symbol can be typed into the field on the form, when the folder is submitted and re-opened, the symbol will not be displayed.

Viewing PDF files

Opening and then closing a PDF file when a PDF action or blank form is displayed causes the form to be blanked out. It is recommended PDF files are not viewed while a PDF action or blank form is open. If you design a procedure that uses PDF forms, ensure that users are aware of this.

Integration Wizard

The three Integration Wizard options in the Form Extensions category (Run a form script, Run a submit script and Run a Cancel script) are not supported for external forms.

7 DOCUMENT MANAGEMENT SUPPORT

Metastorm BPM provides a Document Management Support (DMS) system to enable DMS Providers to integrate with Metastorm BPM. This functionality enables users to interact with Metastorm BPM processes.

Metastorm BPM provides features for Microsoft Office SharePoint Server 2007 and Windows SharePoint Services 3.0 to enable SharePoint to act as a Metastorm BPM client. SharePoint users are able to interact with Metastorm BPM processes.

The supported Document Management Systems are:

- Microsoft Office SharePoint Server 2007

7.1 What does Metastorm DMS provide?

Metastorm DMS provides the following features:

- DMS Clip Field – the standard attachment clip has been extended to support a single DMS document.
- DMS Multiple Attachment Clip field – the multiple attachment clip has been extended to support multiple DMS documents.
- DMS Common Library – a Metastorm library which contains Integration Wizard functions to work with the DMS clip and the DMS multiple attachment clip fields.
- DMS Provider Designer Library – a Metastorm library which contains Integration Wizard functions to enable Process Designers to execute DMS commands in server-side scripts. A DMS Provider Designer Library accompanies each DMS Provider.
- DMS Browser Control – enables users to browse a DMS repository to select a document or location.
- Inherited DMS permissions – the file permissions already assigned in the DMS provider are used by Metastorm DMS.

 © Metastorm Inc., 2008

For Release 7.6 SharePoint is the only DMS Provider supported. To create your own DMS Provider, refer to the DMS Provider Framework documentation.

7.2 Setup

Before using the DMS attachment clip functionality, the following need to be set up and configured:

- Installation Prerequisites – refer to the Installation Prerequisites Guide.
- Document Management Support system installed from the core Metastorm BPM install.
- web.config - needs to be configured for DMS. The Metastorm BPM installer should automatically configure the web.config file.
- Publish DMS libraries.
- Designer Procedure Properties

7.2.1 Procedure Properties

The Procedure Properties dialog has a Document Management tab which must contain the DMS Provider name.

If the DMS Provider name is not entered in the Procedure Properties dialog, error messages for DMS may not be displayed correctly.

1. From the **File** menu, select **Procedure Properties**.
2. Click the **Document Management** tab.
3. Enter the **DMS Provider** name and, as required, the **Default Location**.

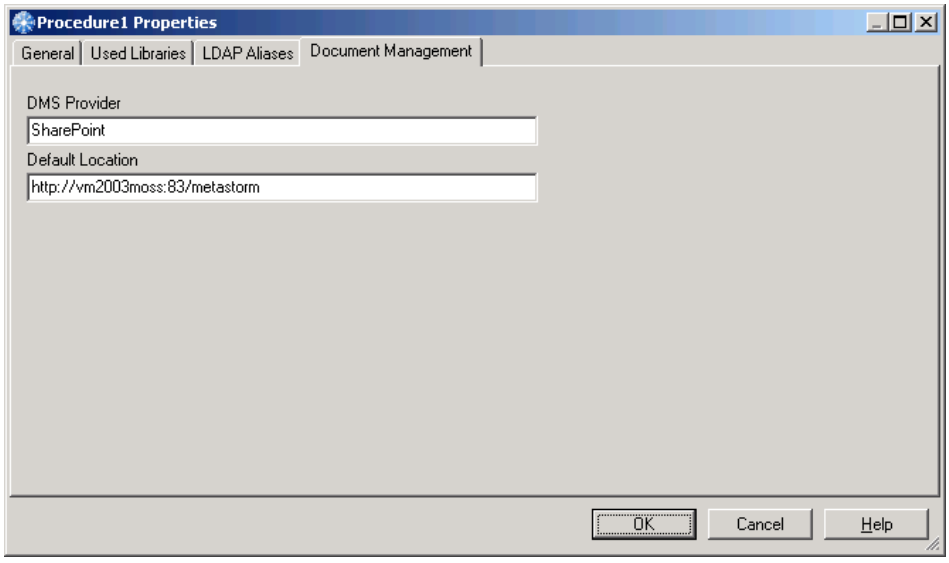


Figure 161: Procedure Properties - Document Management tab

DMS Provider – the name of the DMS Provider. This field is mandatory.

In Release 7.6, SharePoint is the only DMS supported.

Default Location – this is the location of the DMS Provider. Once a default location is defined, Process Users will only be able to access locations under the default location and will not be able to browse to a different location.

DMS Clips created after the default location has been set will automatically have the default location set in the Properties Editor tab.

7.3 DMS Attachment Clips

The Attachment Clip is used to activate DMS clips. There are two types of DMS clips:

- Single DMS Clip
- Multiple DMS Clip

7.3.1 DMS Single Attachment Clip

To create a DMS Single Attachment Clip:

1. Open or create a form.
2. Add an Attachment Clip to the form.
3. In the Attachment Clip Properties, check **Document management support**.

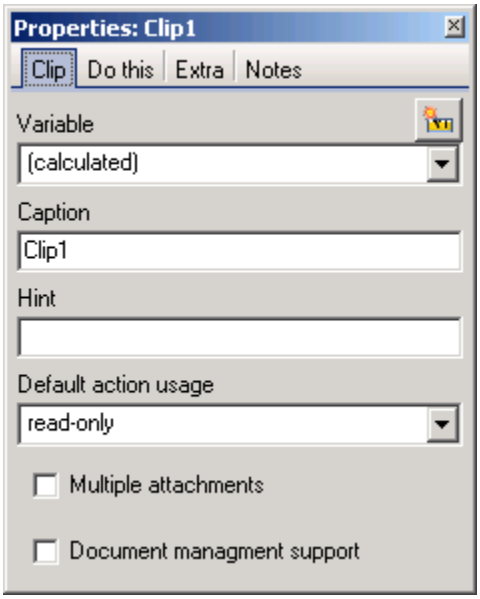


Figure 162: Attachment Clip Properties

The clip changes to a DMS Single Attachment Clip icon.

 © Metastorm Inc., 2008

Figure 163: DMS Single Attachment Clip

4. Create a memo custom variable.
5. Assign the clip to the memo variable.
6. Set the DMS Single Attachment Clip Properties.

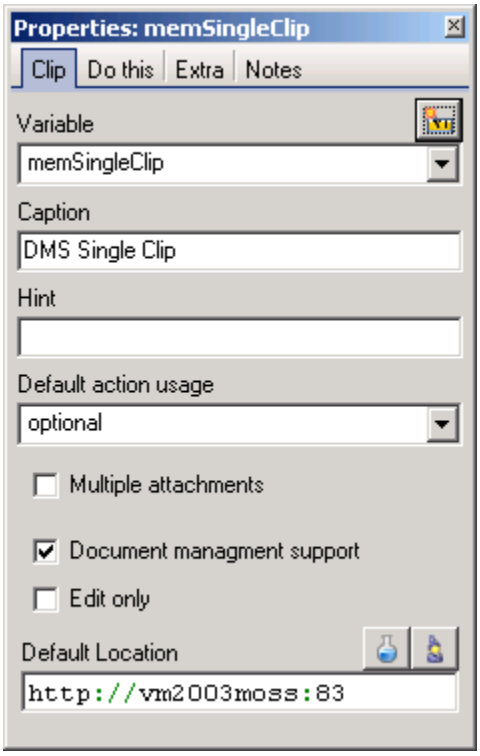


Figure 164: DMS Single Attachment Clip Options

7.3.2 DMS Multiple Attachment Clip

This clip enables users to assign multiple attachments to a folder. The attachments are uploaded individually.

To create a DMS Multiple Attachment Clip:

1. Open or create a form.
2. Click the **Attachment Clip** button.
3. In Attachment Clip Properties, select **Multiple attachments**.
4. In Attachment Clip Properties, select **Document management support**.
5. The Clip changes to a DMS Multiple Attachment Clip icon.

Figure 165: DMS Multiple Attachment Clip

6. Set the DMS Multiple Attachment Clip Properties.

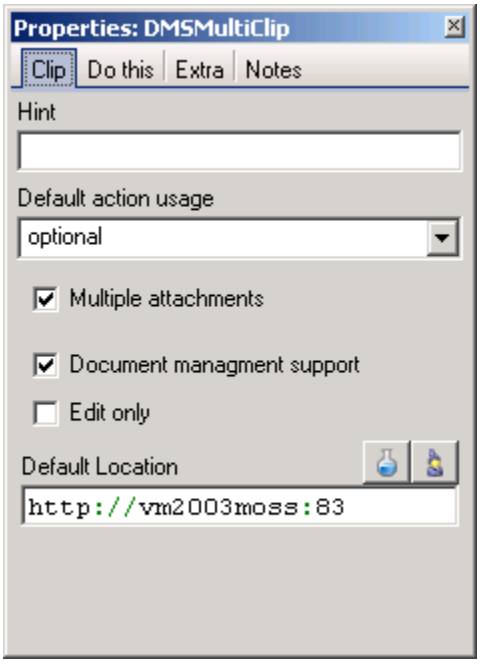


Figure 166: DMS Multiple Attachment Clip Options

7.3.3 DMS Properties

The Properties Editor has the following options in the Clip tab which are relevant to DMS:

Option	*Description*
Multiple attachments	The multiple attachments option is used to create a multi-clip. To create a DMS multi clip, both multiple attachments and Document management support must be checked.
Document management support	This option is checked to enable DMS support.
Edit only	If this option is checked, the Remove and Delete menu options are not available to the Process User. Once a document has been uploaded or selected in a single clip, the option Upload and Select will also be unavailable.
Default Location	This option defines the default location of the DMS Provider.

 © Metastorm Inc., 2008

7.3.4 Dependants

A DMS multiple attachment clip and a DMS single attachment clip can have dependant fields.

Field has dependants of its own is available on the Do This tab. For further information refer to Dependencies section.

7.3.5 Edit Only

This option could be used for different actions on a procedure, for example, create a form with full access to a DMS clip and assign to an action. Then, create another form with the same variable and DMS clip and assign to an action but check Edit Only. Then the review is only able to review and not remove or delete.

7.3.6 Creating a Dynamic Default Location

1. Click on the Formula Editor button .
2. Use the % character to invoke the custom variable context menu to embed dynamic content in the default document location.

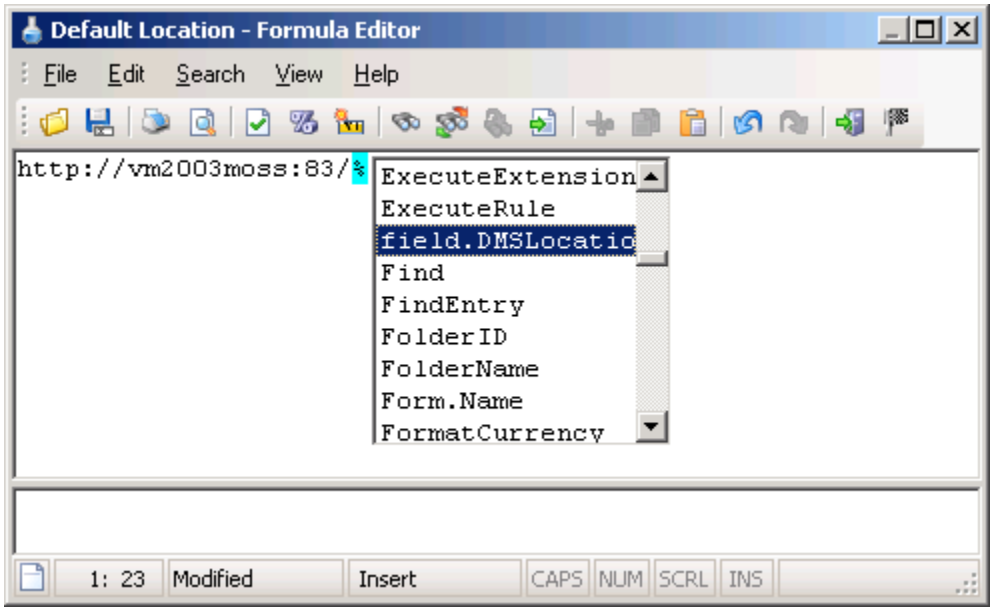


Figure 167: Dynamic Default Location

3. Select the **Text** or **Memo** field name in the context menu to create the dynamic location at runtime.

7.4 Integration Wizard

Each supported DMS provider is shipped with its own Metastorm Designer library containing Integration Wizard functions. These enable Process Designers to execute DMS commands in server-side scripts.

These files can be found in the default location:
`<Metastorm BPM Install>\Administrative Tools\Administrative Procedures`

- `DMS Common Library.xel` – common DMS library
- `SharePoint DMS Library.xel` – library for specific DMS provider

7.4.1 Common DMS Integration Wizard Commands

Get Document Id From Clip

This function retrieves the document Id from a DMS clip. This function only works with the DMS Single Attachment Clip only.

Parameter:

- Variable - select the custom variable which is mapped to the DMS single attachment clip.

Returns:

- Text data type.

Get Selected Document Id

This function retrieves the document Id of the currently selected document in a DMS single attachment clip or a DMS multiple attachment clip field.

Parameter:

- Field Name –the DMS clip field name.

Returns:

- Text data type.

Get Folder Document Ids

This function returns a delimited list of document Ids attached to the current folder.

Parameter:

- List Delimiter – a delimiter to separate the document Id list items.

Returns:

- Memo data type.

This function could be returned into a list field with the list delimiter set to the same delimiter type defined in the function.

7.4.2 SharePoint Integration Wizard Commands

Upload Document

This function uploads a document to the Document Management System.

 © Metastorm Inc., 2008

Parameters:

- Server File Name – the file name as it will be saved on the server.
- Local File – the full path and file name of the file to be uploaded.
- Server Location – the location of the DMS Provider.

Returns:

- Text data type.

Browse SharePoint

This function can browse a specific DMS location.

Parameter:

- SharePoint Location – the URL of a SharePoint Library, Folder or Site.

Returns

- Returns the Document Management System location in an XML fragment as a memo data type.

Check In Document

This function checks a document into the Document Management System.

Parameters:

- File Name – the URL of the file to check in.
- Comments – comments to include with the checked in version of the document.

Returns:

- Integer data type.

Check Out Document

This function checks a document out of the Document Management System.

Parameter:

- File Name – the URL of the file to check out.

Returns:

- Integer data type.

If a document is already checked out a message is displayed to inform the user.

Delete Document

This function deletes a document from the Document Management System.

Parameter:

- File Name - the URL of the file to delete.

Returns:

- Integer data type.

Download Document

This function downloads a document from the Document Management System to the local file system.

Parameters:

- File Name – the URL of the file to download.
- Local File Name – the full path and file name where the document is to be saved on the local file system.

Returns:

- Integer data type.

Get Document Meta Data

This function retrieves Meta Data for the specified document.

Parameters:

- File Name – the URL of the document to query for meta data.
- Meta Data Column – a comma-separated list of metadata field name(s) to retrieve.

Returns:

- Text data type with a semi colon delimiter.

The Meta Data columns that can be used are:

- *Metastorm Default Column Names – the following column names can be used, Title, Modified, ModifiedBy, Created, CreatedBy, CheckedOutTo, CheckInComment,*
- *Custom Column Names – use the display name. The custom column name is case sensitive.*
- *System Column Name – for example, vti_sourceversioncontrol. Refer to the Microsoft SharePoint SDK help for a list of system column names.*

For example: Title;vti_sourceversioncontrol;ModifiedBy

Set Document Meta Data

This function sets Meta Data values for the specified document.

Parameters:

- File Name – the URL of the file to modify.
- Meta Data Column – the meta data column name to modify.

 © Metastorm Inc., 2008

- Variable – select the variable containing the value to be assigned to the meta data column. The variable can contain any data type.

Returns:

- Integer data type.

Refer to Get Document Meta Data for a list of valid column names.

Undo Check Out Document

This function undoes the Document Management System document check out and reverts the document to a pre check out state.

Parameter:

- File Name – the URL of the file to undo from the check out state.

Returns:

- Integer data type.

Verify Document

This function verifies if a specified document exists.

Parameter:

- File name – the URL of the file to verify.

Returns:

- Check data type.

Get Document Author

This function returns the author of the specified document.

Parameter:

- File Name – the URL of the file to return the document author.

Returns:

- Text data type.

Get Document Checked Out By

This function returns the name of the user who has the specified document checked out.

Parameter:

- File Name – the URL of the file to return name of the user who has checked the file out.

Returns:

- Text data type.

Get Document Creation Date

This function returns the date when the specified document was created.

Parameter:

- File Name – the URL of the file to retrieve the creation date.

Returns:

- Date/Time data type.

Get Document Editor

This function returns the name of the user who modified the specified document.

Parameter:

- File Name – the URL of the file which has been modified.

Returns:

- Text data type.

Get Document Last Modified

This function returns the last modified date of the specified document.

Parameter:

- File Name – the URL of the file to return the last modified date.

Returns:

- Date/Time data type.

Get Document Title

This function returns the document title of the specified document.

Parameter:

- File Name – the URL to the file which the title is to be returned.

Returns:

- Text data type.

Verify Folder

This function verifies if a specified folder exists.

 © Metastorm Inc., 2008

Parameter:

- Folder Name – the URL of the folder to verify.

Returns:

- Check data type.

8 DEFINING ROLES

Roles are a way of grouping users. Individuals may be grouped by function, by geographical location, by skill set, or by any other criteria relevant to your organization.

Users are defined as anyone within an organization who will be using the Metastorm BPM system. In most cases, this will include everyone on the staff roster, and may be expanded to include contract or temporary workers, suppliers, customers, etc.

Metastorm BPM uses the concepts of *users* and *roles* to control access to the various parts of the application, and to determine what information users receive from the system. A user can have any number of roles, and a role can be assigned to just one user, or many users. Roles are automatically added to the database when a procedure is published.

The *Metastorm Directory Integration Suite* provides users in a directory services environment a method of gathering user information from an existing directory, while the Metastorm *Users and Roles* tool provides a Metastorm Administrator a simple way to add users, maintain user information, and assign role information to users. Instructions for using these tools may be found in the *Metastorm Administration Guide* and the *Directory Integration Suite* guide.

This section explains how to define roles within a procedure.

8.1 Defining a Role

When designing a procedure, roles are used to decide whether:

- Specific folders should appear on a user's To Do list, Watch list or neither.
- A user is permitted to undertake a particular action.
- A user is permitted to view a form in a folder.
- A specific form will appear on a user's Blank Forms list.

For example, "Credit Manager," "Shipping Clerk," and "Operations Controller" are all roles which may have responsibility for particular stages and actions in the procedure. Using roles rather than assigning particular individuals to the task makes it much easier to maintain a

 © Metastorm Inc., 2008

procedure. As individuals join and leave the organization or change duties within the organization, the administrator has only to reassign the role to another user rather than ask the procedure designer to change the procedure.

Roles are defined through the *Roles dialog*, which may be accessed by clicking on the **Roles** button on the General toolbar, or by selecting the menu option **View | Roles**.

There are two types of role:

- Static

  A role to which users are assigned through the *Users and Roles* tool – for example, "area managers." The same user holds the role for every Metastorm procedure.

  A static role defined through the Roles dialog is added to the database when the procedure containing it is published. Once the procedure is published, this static role may be managed through the *Users and Roles* tool.

  As new static roles are added to the database, their role assignments become available to Metastorm procedures. In order to utilize a static role assignment, the role must be defined in the procedure that uses it.

  It is a good idea to make a master list of the static roles in your organization, and post it where you can refer to it whenever you are designing or modifying a procedure. This list will help prevent you from using different naming conventions for the same role in different procedures, which could result in a number of redundant roles. (e.g. the names "Marketing Director," "Mktg. Director," "Marketing Dir," and "Mktg Dir" all refer to the same role, but each would have to be assigned separately through the Users and Roles *tool.)*

- Dynamic

  A role assignment evaluated in the context of a particular folder when the procedure is processed – for example, the originator of a folder. Dynamic roles may be based on formulas, database queries, content cached from directories, etc.

  The following dynamic roles are set up automatically by Designer:

  - *Everybody* is used to specify all users registered in the database.

For further information on using the 'everybody' role, refer to section 8.1.8, Using the 'everybody' role.

  - *Metastorm Administrator* is used by individuals with authority to administer the Metastorm system, and should be assigned to at least one registered user. This role may also be used by the process designer when creating and testing a procedure.
  - *Originator* identifies the user who created a folder.
  - *Access* identifies all users who hold the role identified in each map's *Limit access to* property.
  - *Metastorm Guest* is used to allow access by anonymous users.
  - *To Do list* is used to indicate all users on whose To Do lists a folder currently sits.
  - *Watch list* is used to indicate all users on whose Watch lists a folder currently sits.

  Dynamic roles are specific to the procedure they were created for, and cannot be managed through the *Users and Roles* tool.

8.1.1 Identifying the Roles in Your Organization

Because role-based work management is based on the concept that responsibilities are assigned based on an individual's job function—or *role*—within the organization, the simplest way to begin identifying those roles is to study your company's organizational chart. While you will seldom use all available roles in any single procedure, developing a master list of the static roles within your organization will minimize redundancy within your Metastorm database. It will also save you the effort of having to define a new set of role assignments through the *Users and Roles* tool each time you create a new procedure.

Your initial role list may primarily consist of job titles or job functions. Notice that when you have more than one individual with the same job title (e.g. Customer Service Representative) that a single role definition will apply to each.

Next study your organizational chart to see what group roles should also be defined. For example, the "Sales Manager" is also a member of the "Sales" group, as well as a member of the generic group "Managers." A contract employee is a member of the generic group "Contractors" as well as a member of a particular team. Work through your list to determine what group role definitions are most commonly used within your organization, and add these to your list.

Finally, consider your company's *unwritten* organization. Is Susan from Customer Service on indefinite loan to the Sales Department? Does John from Production happen to have a cousin in the retail industry, and therefore act as the unofficial Purchasing Agent for your regional office? Is your Office Manager also cross-training as a Technical Writer? Does your company have job-sharing policies in effect that should be considered? As you evaluate these unwritten roles, you will continue to refine your list.

Keep in mind that the list you develop may not reflect all possible roles in your company. However, you may define new roles as you design procedures that require them.

Assigning users to static roles is done through the Users and Roles *tool. Instructions for using this utility may be found in the* Metastorm Administration Guide.

8.1.2 User Attributes and Relationships

Dynamic roles are assigned through a procedure by using Metastorm functions to identify a set of attributes that the holders of the role must have. These attributes are often more easily specified by the relationship certain attribute sets have with each other.

Attributes

Attributes allow administrators to categorize users based on one or more categories that help to define that user. For example, attributes might consist of Department, Title, Extension, Fax Number, etc.

 © Metastorm Inc., 2008

Each attribute can be assigned one or more corresponding values that further define the attribute. For example, the Department attribute values might be Marketing, Sales, Development, and Personnel, as illustrated in the following table:

User	*Attribute*	*Value*
Kim	Department	Marketing
John	Department	Sales
Michael	Department	Development
Susan	Department	Personnel

Table 29: Metastorm Attributes Table

Multi-level attributes can be assigned to one person (for example, Kim can be the Director of Sales and the Manager of Marketing).

Relationships

Relationships can be defined through the Metastorm *Users and Roles* tool and the Metastorm *Directory Integration Suite* tools. Relationships are composed of groupings of attributes, and, in a sense, "tie" attributes together.

For example, a company administrator may set up an attribute called "Title" and an attribute called "Department." These attributes will be used to differentiate employees by title and department. For each attribute, one or more values may be assigned. The following table shows possible values for the Title and Department attributes:

Title	*Department*
Director	Marketing
Manager	Sales
Team Lead	Public Relations

Table 30: Attributes and Values

Let's assume there's an employee named "Kim" who has the attributes shown in the following table:

Kim
Title: Director
Title: Manager
department: Sales
department: Marketing

Table 31: Employee Attributes

We know from these attributes that Kim has a double role as a manager and director—but is she the manager of Marketing or of Sales? Her attributes, as listed here, don't give us that information. However, if we create a *relationship* called "position" that ties together the Title attribute with her corresponding Department attributes, we will understand what her functions are

and how the attributes assigned to her relate to one another. The following table shows the structure of the relationship "Position" as it relates to Kim:

Name	Title	Department
Kim	Director	Marketing
Kim	Manager	Sales

Table 32: Sample Attribute Structure

Note that the same user, Kim, can have more than one value (Sales, Marketing) assigned to the same attribute (Department). See "Attributes" above.

Once this information is stored in a database, the data can be used by in a procedure.

8.1.3 Defining Roles Using Metastorm Data Extraction Utilities

The *Metastorm Directory Integration Suite* data extraction utilities are provided with Metastorm BPM, and offer users the ability to extract user and attribute information stored in one of the following directory services environments and cache this information in the database:

- **NDAP** – for accessing Novell eDirectory data.
- **RDBMS** – for accessing Relational Database Management System data.
- **LDAP** – for accessing Lightweight Directory Access Protocol (LDAP), Microsoft Active Directory data, and other generic protocol data.

In addition, the *Users and Roles Administration Services, Microsoft Active Directory Property Page,* has been created, which allows you to define custom attributes within Novell eDirectory and Microsoft Active Directory. These attributes are also exported by the *Directory Extraction Service* and cached within the database for use by Metastorm BPM.

Instructions for using these utilities are found in the Metastorm Administration Guide.

The user information extracted through the *Directory Extraction Service* can be used to define roles in two ways:

- **Dynamic roles** – user attributes imported through a *Directory Integration Suite* utility may be used in the formulas used to define dynamic roles. This will be discussed in greater detail later in this section.
- **Static roles** – users imported into the database through a *Directory Integration Suite* utility may be assigned static roles through the Users and Roles tool. Instructions for using this utility may be found in the Metastorm Administration Guide.

8.1.4 Adding Roles to the Roles List

To add roles to the roles list, first open the procedure whose roles you wish to modify.

1. From the **General toolbar**, click on the **Roles** button . Alternatively, select **View | Roles** from the menu.
    - The *Roles* dialog is displayed.

 © Metastorm Inc., 2008

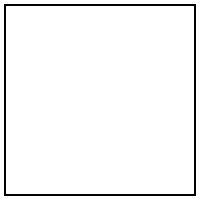


Figure 168: Roles dialog as an independent display

Dynamic roles are identified by a formula defined in the **Formula** column.

The default roles created by Designer are not assigned to any particular map, and appear on the Common *Roles tab. Roles created for and assigned to specific maps will be placed on tabs corresponding to the map to which they are assigned.*

2. The Role dialog can also be called from the **Common** tab of the procedure map by clicking on the Roles icon.

3. Roles can be added and removed. To add a Role click on the Add icon on the dialog toolbar, or right click in the Role dialog and select the **Add** icon.

    - A new line is added to the bottom of the list of current Roles.

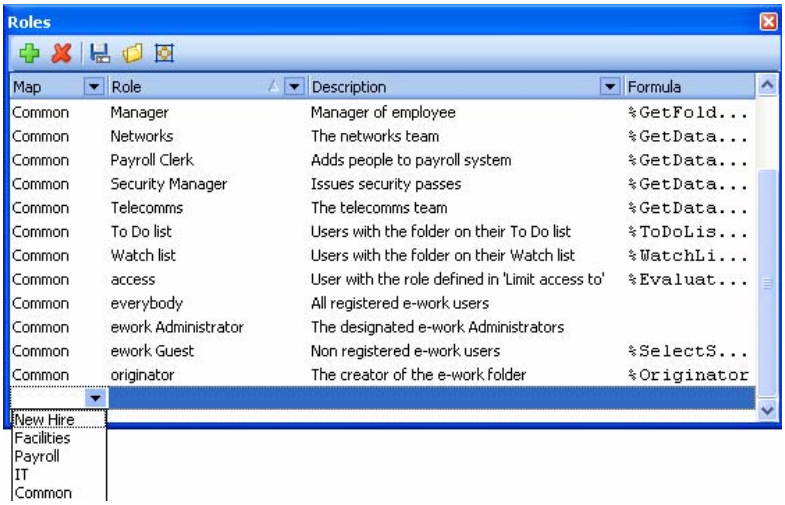

Roles
Map
Role
Description
Formula
Common Manager Manager of employee %GetFold...
Common Networks The networks team %GetData...
Common Payroll Clerk Adds people to payroll system %GetData...
Common Security Manager Issues security passes %GetData...
Common Telecomms The telecomms team %GetData...
Common To Do list Users with the folder on their To Do list %ToDoLis...
Common Watch list Users with the folder on their Watch list %WatchLi...
Common access User with the role defined in 'Limit access to' %Evaluat...
Common everybody All registered e-work users
Common ework Administrator The designated e-work Administrators
Common ework Guest Non registered e-work users %SelectS...
Common originator The creator of the e-work folder %Originator
New Hire
Facilities
Payroll
IT
Common

Figure 169 : Role Entry

4. From the dropdown list of maps, see *Figure 169* above:

    - Assign the role to the common roles list, select **Common** from the drop-down list.
    - Assign the role to a specific map such as Flight, select that map.

5. Enter role name and a description in the **Role** and **Description** fields. A role name cannot be longer than 31 characters, and may only include letters (A-Z or a-z), numbers (0-9), underscores and spaces.

Non US English characters and extended characters may not be used in a role name.

A role name cannot duplicate the name of a user defined in the Users and Roles utility.

6. If you need to enter a formula, you can do so by:

    - Typing in the formula in the space on the line entry;
    - Click on the **Formula editor** button and create the formula through the Formula editor; or
    - Click on the **Integration wizard** button and create the formula through the Integration wizard.

When a formula is being defined, only custom variables from the selected map or sub-map may be used. When adding a text field to a form in a sub-map care must be taken if the procedure has not been published. If the IW is invoked and Folder information is selected followed by Get Child Text, the custom variables displayed are those of the parent map, not the Child map, as expected

8.1.5 Deleting Roles from the Roles List

To delete roles from the roles list, first open the procedure whose roles you wish to modify.

1. From the **General toolbar**, click on the **Roles** button . Alternatively, select **View | Roles** from the menu.
    - The *Roles* dialog appears.
2. Select the Role in the Roles list and then click the **Delete** button , press the ***Delete*** key on your keyboard, or right click in the Role dialog and select the **Delete** icon.
    - The Role is deleted from the list.

No warning is given that the role is about to be deleted.

You cannot delete the default roles (Access, Everybody, Metastorm Administrator, Metastorm Guest, Originator, To Do list, Watch list). When these are selected the delete button is disabled.

8.1.6 Editing Roles

To edit roles, first open the procedure whose roles you wish to modify.

You cannot *edit the default roles created by Designer.*

1. Select the role to be modified in the list.
2. Double click the selected role.
    - A Role dialog box is displayed.

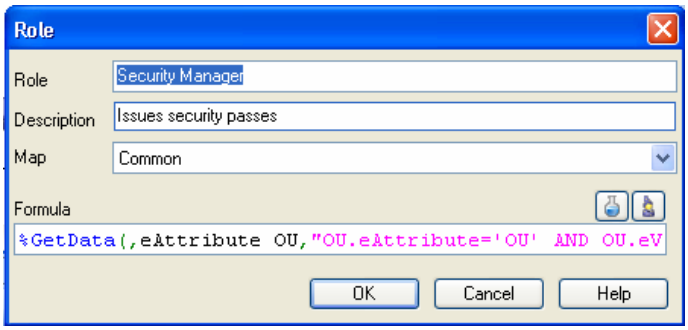


Figure 170: Role modification dialog

3. Modify the role name and a description in the **Role** and **Description** fields.
4. From the **Map** drop-down, you have these choices:
   - To reassign the role to the common roles list, select **Common**.
   - Select a map to reassign the role to a specific map.
5. If you need to edit the **Formula**, you can do so by:
   - Typing in the formula; or
   - Clicking on the **Formula editor** button; or
   - Clicking on the **Integration wizard** button.

Detailed instructions on how to use the Formula editor and the Integration wizard may be found in Section 10, "Creating Formulas", page 261.

6. Click **OK** to confirm the changes and close the dialog.

Editing a role name to is the same as deleting the old role name and adding a new role. If you edit a role name, you should confirm that your role assignments are still in force.

8.1.7 Defining Dynamic Role Formulas

Dynamic roles are defined by a formula. The formula entered should resolve to a list of user names when evaluated at run-time.

For further information on creating formulas, refer to Section 10 Creating Formulas and the Appendix on Formula Components.

Using operators within dynamic role formulas

Formulas created for dynamic roles are treated as literal strings. In general, functions and variables that begin with '%' will be evaluated as expected. However, if the formula contains operators (and these are not intended to be treated as literal strings), the formula should be wrapped in a %Evaluate() function.

The operators are:

```
? + * / -
```

Below are two examples of possible dynamic role formulas. For this example, assume that:

```
%exampleVar1 = "Paul Pascal"

%exampleVar2 = "Sarah Singalong"

%exampleVar3 = false
```

Example One

In this example, the formula contains no operators.

```
%GetData(exampleVar1), %GetData(exampleVar2)
```

The data returned from this formula is:

“Paul Pascal, Sarah Singalong”

Example Two

In this example the formula contains the operator ‘?’.

```
(%exampleVar3)?(%exampleVar1):(%exampleVar2)
```

The data returned from this formula is the string:

“(false)?(Paul Pascal):(Sarah Singalong)”

In order to have the formula evaluated, it must be wrapped in a %Evaluate() function, as follows:

```
%Evaluate((%exampleVar3)?(%exampleVar1):(%exampleVar2))
```

The data returned from this evaluated formula is:

“Sarah Singalong”

Using %LDAPSearch within dynamic role formulas

The `%LDAPSearch` function returns attribute values of zero or more directory entries that match the specified search filter. Each attribute is separated with a new line character, which is the format required to fill lists and memo fields on forms.

Process Engine role lists must be comma-separated and the presence of a new line character will invalidate the role evaluation so no users will appear to be on the list.

If `%LDAPSearch` results are to be used in a dynamic role formula, they must be reformatted using `%Replace` to produce a valid comma-separated role list. For example:

```
%Replace(%LDAPSearch(<alias>,<filter>,<attribute>,0),%newline(),"
,",0)
```

8.1.8 Using the ‘everybody’ role

If you are designing a procedure for a large number of users, we recommend that you do not give all the users access to a stage or an action using the ‘everybody’ role.

 © Metastorm Inc., 2008

All roles are published as formulas that return a list of usernames. The ‘everybody’ role is published as the following formula:

```
%GetData(,eAssignment,"(eRoleName='everybody')","eUserName"),
```

This formula returns a list of usernames of users who have the ‘everybody’ role assigned to them, i.e. all users.

When a folder is created, the role formula is evaluated to determine whose To Do and Watch lists the folder should appear on. A new alert is created for each user whose username is returned by the formula. If the role is set to ‘everybody’, an alert is created for each user.

When a folder is opened, the role formula for each action is evaluated to determine whether the action is available to the current user. If the current user name exists in the resulting list, the user is given access to the action.

In a setup with a large number of users, this is not efficient. We recommend that you do not allow stages or actions to be accessible to all users. However, if this cannot be avoided, the following workaround can be applied.

Workaround

Continue to use the ‘everybody’ role to add a folder to all To Do or Watch lists.

To allow all users access to an action, we suggest you create and use a new role called ‘anybody’. Assign the following formula to the role:

```
%User.Name
```

When this formula is evaluated it will return the username of the current user and consequently avoid the need to search a long list of usernames.

9 DEFINING FLAGS

Flags are a trigger mechanism used by procedures to allow communication:

- From one map to one or more other maps within the same procedure.
- From a map in one procedure to one or more other maps in a different procedure.
- Via ActiveX controls or the command line from an external application to one or more maps in one or more procedures.

Flags may be used in conjunction with Flagged Actions to trigger updates to one or more existing folders, or to create new folders (as in the case of sub-procedures).

When a flag is raised, one or more data items may also be passed to each folder that is waiting for that flag. This data is known as *Flag Data*. Flags and the flag data they pass are defined through the *Flags dialog*.

This section explains how to create flags and define flag data.

9.1 Flags

Flags may be raised within a procedure (internal flags), or by the use of commands external to the procedure (external flags).

When a flag is associated with an action, the specified flag is triggered when a folder meets the procedure's criteria. It is also possible, although much less common, to create a formula to raise a flag from a stage.

Flags may be associated with the appropriate action in the following ways:

- Raised through an action's **Raise Flag** property.
- Raised through formulas in the **Do this** tab of Properties editor.
- Raised automatically as part of a sub-procedure stage and used to open another map, but with no sharing of data.
- Raised externally by the use of the eraiseflag.exe utility.
- Assigned externally by the use of the eraiseflag.ocx component. (Refer to the SDK for instructions).

 © Metastorm Inc., 2008

- Raised by a Transaction Protocol request. (Refer to the SDK for instructions).

9.2 Defining a Flag

9.2.1 Flag Data

Flag data is the term used to identify the information that a flag passes. In order to pass flag data, you must do the following:

- Assign a value to the "flag data" field. The receiving procedure sees this information as %Session.FlagData[1], %Session.FlagData[2], %Session.FlagData[3], etc., rather than by the sending procedure's variable name, so it's important to keep track of the order in which this data is requested.
- Assign variables in the same order on the receiving end (i.e., in a submap).

In order to eliminate confusion, create variables of the SAME name in both maps (e.g. FirstName, LastName, in the sending procedure and FirstName, LastName in the receiving procedure) and assign both variables to the flag data to be passed.

- In a child/parent relationship, if you want a child folder to share data with the parent, you can pass data in a flag. However, if there is a large amount of data, this is not the most efficient method. In these instances, it is more common to use a formula (placed in a field on the **Do This** tab of the Properties editor) to query the database directly.

9.2.2 Flags Dialog

When you are designing a procedure, you can access the Flags dialog from the menu (**View | Flags**) or by clicking on the **Flags** button on the General toolbar.

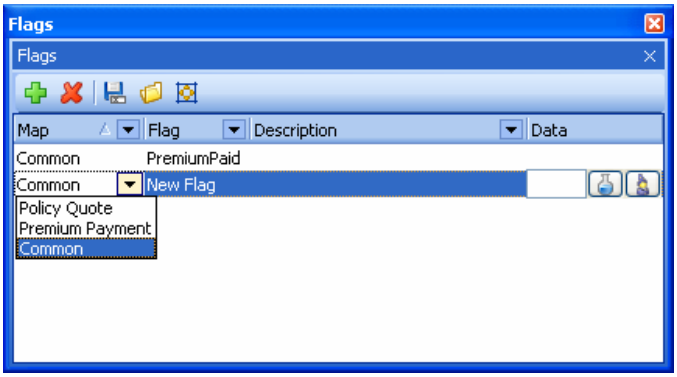


Figure 171: Flags dialog

1. You can add flags by either of the following methods:
   - Click the add icon on the Flags dialog, or right click the mouse and select the **Add** icon from the pop-up menu.
   - Press the **Insert** button on your keyboard.
   - Each of these options adds a new line to the list of flags.

2. If the flag is to pass values from custom variables when it is raised, specify the map to which the custom variables belong by clicking in the **Map** field in the Flags dialog and selecting from the drop-down list. All maps in the current procedure are listed. Custom variable data may be passed only from the current map.

3. Enter the name for this flag in the **Flag** field. A flag name cannot be longer than 31 characters, and may only include letters (A-Z or a-z), numbers (0-9), underscores, hyphens and spaces. Non US English characters cannot be used in flag names.

Whenever possible, try to use unique flag names. This avoids multiple maps with shared flags. Remember, flags will affect all *published procedures, so procedures that unintentionally share a flag may cause unexpected results.*

If you have a flag name and definition in the sending procedure, you need a flag of the same name defined in the receiving procedure in order to use the flag to activate a flagged action in the receiving procedure.

4. Enter a brief **Description** of the flag. This description is useful when you are using several flags in a procedure or are using the same flag across several procedures.

5. If the flag is to be raised using the **Raise Flag** property (rather than `%RaiseFlag`), define the values to be passed by defining a formula in the **Data** field. Click on either the **Integration wizard** or **Formula editor** buttons to use one of these tools to define the formula, or you may type the formula directly into the field.

6. Click on **OK** to save.

Each defined flag is available for selection in a Flag action, as well as by an action in any map in the procedure.

9.2.3 Flags Dialog Buttons

Other buttons on the Flags dialog allow you to perform the following tasks:

Figure 172: Flags Toolbar

- **Delete** button – Allows you delete the flag (with confirmation). Be aware that removing the flag must be reflected in any stages or actions associated with this action in any affected procedures.
- **Open** – Opens a standard File Open dialog, allowing you to open a previously saved Flags file.
- **Save As** – Opens a standard File Save dialog, allowing you to save the current Flags file.

 © Metastorm Inc., 2008

- **Best Fit** – Resizes the columns for optimum display of the contents.

*The **Open** and **Save As** options are particularly useful when you need to create the same flags in multiple procedures.*

To **Edit** a Flag – Select an entry in the Flag dialog box and double click.

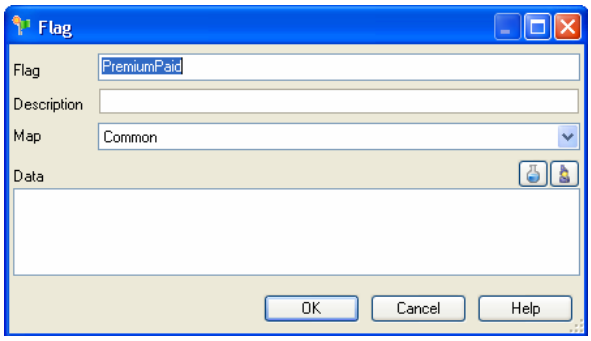


Figure 173: Edit Flag details

Opens the edit Flags dialog, allowing you to edit the flag name, flag data, map the flag is assigned to, etc. Be aware that any modifications you make to the flag must be reflected in the procedures that are affected.

9.3 Raising Flags from within Procedures

9.3.1 Properties editor

You may associate a flag with an action from the action's Properties editor using one of the following methods:

1. On the **Action** tab, select the flag from the drop-down list of previously defined flags available in the **Raise Flag** property.

   OR

2. On the **Do This** tab, specify the formula for the flag in the **When action started** property by directly entering the formula or by using the property's Integration wizard or Formula editor.

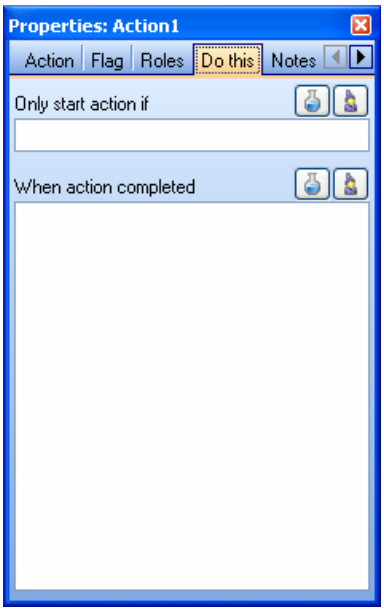


Figure 174: Do This Tab

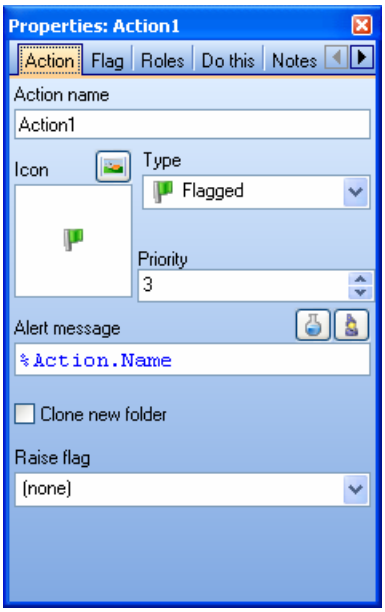


Figure 175: Action Tab

9.3.2 Using eraiseflag.exe to raise a flag from an external application

The program *eRaiseFlag.exe* can be used by any external application or any user to raise a flag outside of Metastorm BPM. If you need to raise more than one flag at a time, you can create a *.bat* file to automate this command line tool. The syntax for *eRaiseFlag.exe* is as follows:

```
eRaiseFlag.exe /FlagName:<name> /FlagFolder:<folder id>
/FlagData:<flag data> /ServerName:<server name>
```

Switches in *eRaiseFlag.exe* are defined as follows:

/FlagName

This switch is required to select eRaiseFlag.exe. This is where you define the name of the flag you want to raise.

For example: `eRaiseFlag.exe /FlagName:firstname1`

This will raise the flag named “firstname1.”

/FlagFolder

You may specify a folder in order to move only that folder.

For example: `eRaiseFlag.exe /FlagName:firstname1 FlagFolder:00000000000000000000000000000034`

This will raise the flag “firstname1” and affect only the folder with the folder ID of “00000000000000000000000000000034”.

 © Metastorm Inc., 2008

If the folder ID exceeds 31 characters, errors will be displayed and the folder will not be moved.

/FlagData

You may define any flag data to be passed along with the flag. If more than one piece of flag data is to be passed they are delimited with tabs. If you are passing multiple amounts of flag data, the entire switch must be encapsulated in double quotes.

For example: `eRaiseFlag.exe /FlagName:firstname1 "/FlagData:data1 data2 data3"`

This will raise the flag "firstname1" with three flag data items: "data1," "data2," and "data3."

/ServerName

If you are running *eRaiseFlag.exe* on a machine other than that of the Process Engine, you must specify the computer name of the system running the Process Engine. In order to raise a flag from one machine to another you must have appropriate DCOM rights to the server running the Engine.

For information about DCOM rights to the Process Engine server, refer to the Administration Guide.

For example: `eRaiseFlag.exe /FlagName:firstname1 /ServerName:metastormbpm`

This would raise the flag "firstname1" on the server named "metastormbpm."

/?

The syntax of the command may be displayed using the /? option.

For example: `eRaiseFlag.exe /?`

9.3.3 Receiving Flag Data

For the receiving procedure to use flag data, it must be able to capture it. To do this, you must define the flag data as a custom variable in the Flagged Action stage that is to receive the data:

1. Select the appropriate Flagged Action, and open the Properties editor.
2. Select the **Do this** tab.
3. Click on the **Integration wizard** button adjacent to the **When action completed** field to open the Integration wizard.
4. From the category **Commands**, select the item **Assign value**.
5. Select the **Variable** from the drop-down list. If the variable has not yet been created for this map, click on the **Custom Variable** button to create it. The variable will be added to the drop-down list, and you may then select it.
6. In the **Value** field type:

   `Session.FlagData[x]`

   (where x is the position of the flag data)

   for each flag data item you wish `this` flagged action to receive.

- Click on the **Integration wizard** button to open the Integration wizard.
- From the category **Action and Form Information**, select the item **Flag data**.
- You will be prompted for the position of the flag data. Select the appropriate position from the drop-down list and press finish twice to return to the Property editor.

The formula now in the **When action completed** field will allow the receiving map to capture the flag data.

10 CREATING FORMULAS

Much of the functionality of Metastorm BPM is provided by the formulas that direct the action of procedure components. These formulas may be simple or complex, and may be created using tools provided within the Designer.

In this section, we will introduce the following formula-building tools:

- Integration wizard
- Condition Builder
- Formula editor

This section also discusses the following topics, which are useful in formula creation:

- Accessing Databases
- Accessing Directories
- Accessing Business Rules Engines
- Custom Variables
- Table Editor

Finally, the section contains a glossary of Integration wizard items (grouped by category), to assist process designers to make the most effective use of this tool.

10.1 Integration Wizard

When you are defining an element on a form or a map, you use the Properties editor to specify the properties of the element. For some of these properties you may wish to use the Integration wizard to enter the formulas that the Process Engine evaluates at run-time.

The Integration wizard was created to provide easy integration to external systems such as email and databases, and easy access to Metastorm functions and variables. It has built-in intelligence so that it prompts only with information that is appropriate for the context.

The main functions of the Integration wizard are to provide:

- Easy integration to systems that are external to Metastorm BPM.

- Easy access to variables and functions that may be used in formulas.

By allowing you to integrate to systems that are external to Metastorm BPM, the Integration wizard enables you to specify that a procedure can, for example, print a Word document, send an email, read from a database, and run applications.

The Integration wizard is a tool for building expressions and calculations based on system and custom variables and functions built into the Metastorm BPM system. The Integration wizard may be opened at any time to create or edit your formula by clicking on its button:

Figure 176: Integration wizard Button

The Integration wizard provides a step-by-step interface, allowing you to select from a list of formula categories and a subsequent list of items for each category that are valid for the property field you are working with. Each property has a related list of valid formula types; only items of compatible types will be displayed in the formula list.

You can also open the Integration wizard to define a formula when defining flag data (using the *Flags* dialog) or roles (using the *Roles* dialog).

When you click on the **Integration wizard** button, Designer displays the *Integration Wizard* dialog to help you build the formula.

The resulting formulas are displayed with the following color-coding:

- Plain text, which will not be translated by the Process Engine at run-time, is shown in black or magenta for quoted strings
- Variable and function names are shown in blue
- Function argument delimiters (parentheses and commas) and arithmetic and logical operators are shown in green.

You can also edit the formula directly, without using the Integration wizard. You may use the Formula editor to make any further changes required.

After you have made your selections from the available formula types, the Integration wizard builds the expression for you and places it into the property field you selected.

The Integration wizard provides a list of categories and a list of items for each category. If a property already contains a formula, then the relevant category and item will be selected when the Integration wizard is opened. Otherwise, each property has its own default category that will be selected.

Each property also has an expected data type; only items of compatible types will be displayed by default (to show other items, grayed out, uncheck the **Hide invalid check box** above the item list).

Variable names and function names are always preceded by a % sign. When editing a formula outside of the Integration wizard, the percent sign must be included.

 © Metastorm Inc., 2008

When editing formulas, you have the following options:

- To add a new line to the formula, add a blank line and then click on **Next**.
- To edit an existing line, select it and then click **Next**.
- To delete an existing value, select it and then click **Delete**.

To reorder the lines of the formula, select a line and click the **Up** or **Down** buttons in the Integration Wizard dialog.

10.1.1 The Integration Wizard Interface

If you start the Integration wizard when your cursor is positioned in a property that contains no commands, Designer displays the main **Integration wizard** dialog with a list of all categories. The relevant category and available items within that category are displayed. The main Integration wizard dialog box contains the following components, as illustrated:

Figure 177 Integration Wizard Screen Components

The following table lists the Integration Wizard components:

Component	*Purpose*
Title Bar	This displays the name of the property for which you are setting the formulas.
Hide invalid check box	If this box is checked, Designer will hide any category or item options that do not apply to the selected procedure component.
Category list	This displays a list of the categories available for the selected property. When you select a different category, Designer updates the items displayed in the Item list.
Item list	This displays in the selected category a list of the items that are compatible with the type of property being set. This list will vary depending on the type of property being defined.
Hint	This describes the usage of the selected item.
Back button	Click on this to return to the previous step.
Next button	Click on this when you have made your selection to move to the next step.
Finish button	Click on this when you have finished using the Integration wizard to change the property's formula.
Cancel button	Click on this to leave the Integration wizard and leave the property's formula unchanged.

Component	Purpose
Help button	This provides help for the Integration wizard.

Table 33: Integration wizard Component Functionality

10.1.2 Using the Integration Wizard to Create a New Formula

1. From within your Metastorm procedure, select the component you wish to work with.
2. From the Property **editor** for the selected component, select the **Integration wizard** button.

    - The Integration wizard will open:

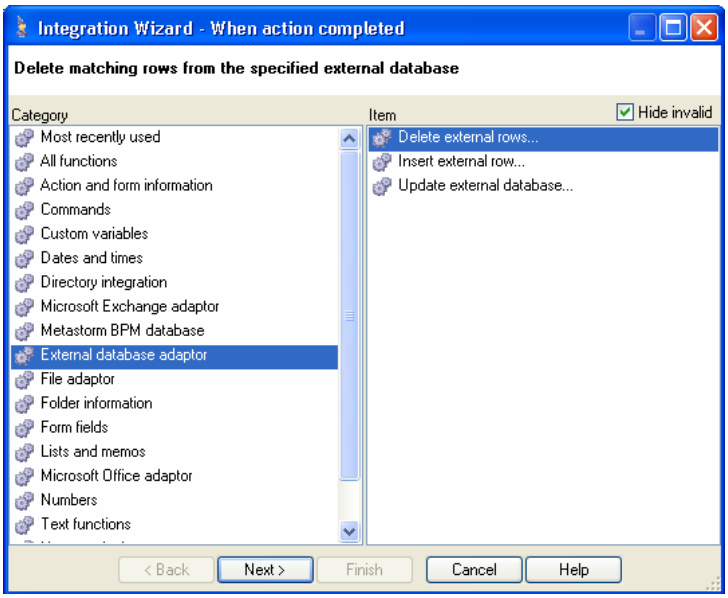


Figure 178: Integration wizard

3. Select a category from the available categories listed in the left column.
4. Select an item from the available items listed in the right column.

If the Hide Invalid *box is checked, only valid categories and items for the selected component will be displayed. Otherwise, all options will be displayed, but unavailable options will be grayed-out.*

A Glossary of Integration wizard Categories and Items, together with brief descriptions of their functionality, may be found at the end of this section.

5. Click on the **Next** button (or double-click on the item). Designer opens the relevant Integration wizard dialog. The dialog displayed will vary, depending on the type of item you have selected. For example, using the options illustrated above, you may select either of two items:

    - Assign value
    - Conditional operation

Each of these selections will be briefly illustrated on the following pages.

 © Metastorm Inc., 2008

Integration Wizard Example: Assign Value

In this example, we selected the variable "Deadline," and will use multiple instances of the Integration wizard to define the deadline.

1. Select an action from your procedure map.
2. Go to the **Do this** tab of the Properties editor:

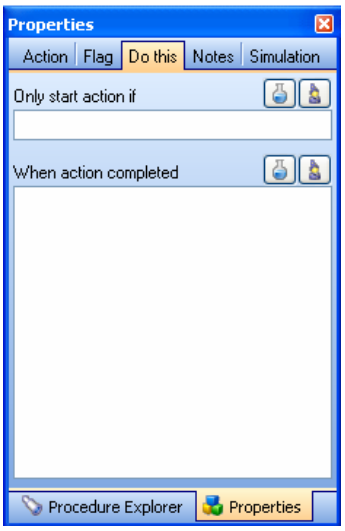


Figure 179: Properties editor- Do this tab

3. From the **When action started** field, click on the **Integration wizard** button.
    - The Integration wizard will open.

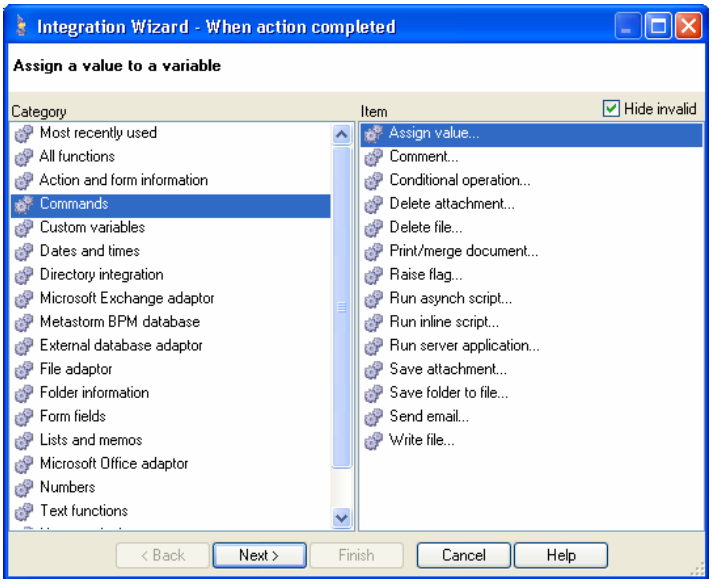


Figure 180: Integration wizard – Assign Value

4. In the Integration wizard from the **Category** column, select Commands.

5. From the **Item** column, select Assign *Value*. Click on **Next** to continue.

    - A new window will open. Select the **Variable** *Deadline* from the drop-down list available from this field.

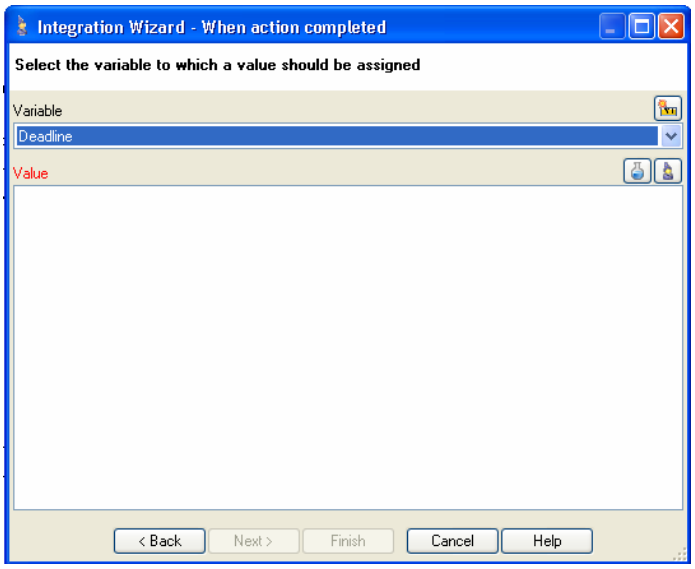


Figure 181: Select Variable Type

6. To define the value, click on the **Integration wizard** button next to the Value field. Designer will open a new (nested) instance of the Integration wizard:

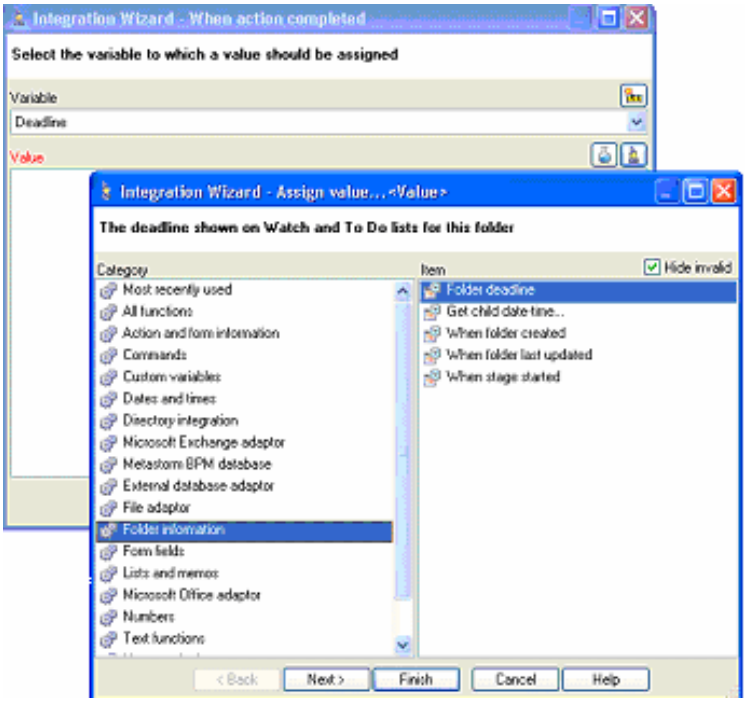


Figure 182: Nested instances of integration wizard

 © Metastorm Inc., 2008

7. From the **Category** column, select *Dates and times*.
8. From the **Item** column, select *Calculate date-time*. Click on **Next** to continue.
    - The Assign Value dialog is displayed.

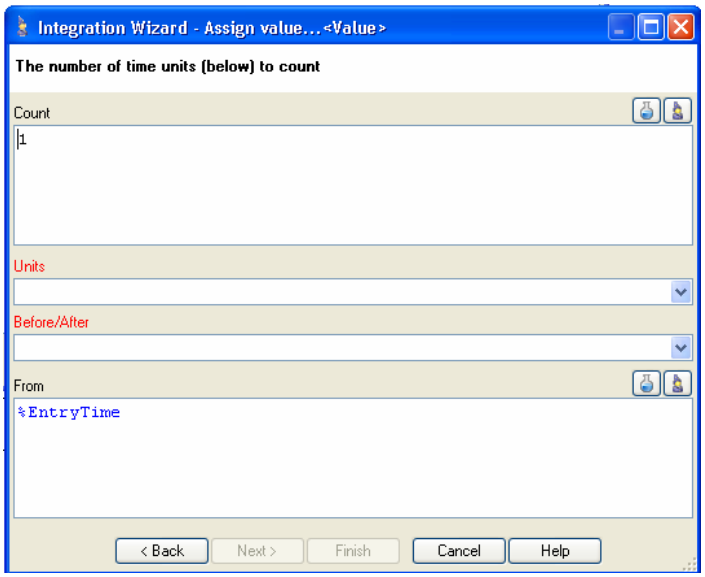


Figure 183: Calculate Deadline

9. Define how the deadline is to be calculated and click **Next.**
    - The expression that has been defined is displayed in the original instance of the wizard.
    - You can open the Integration wizard again from this field, if necessary.

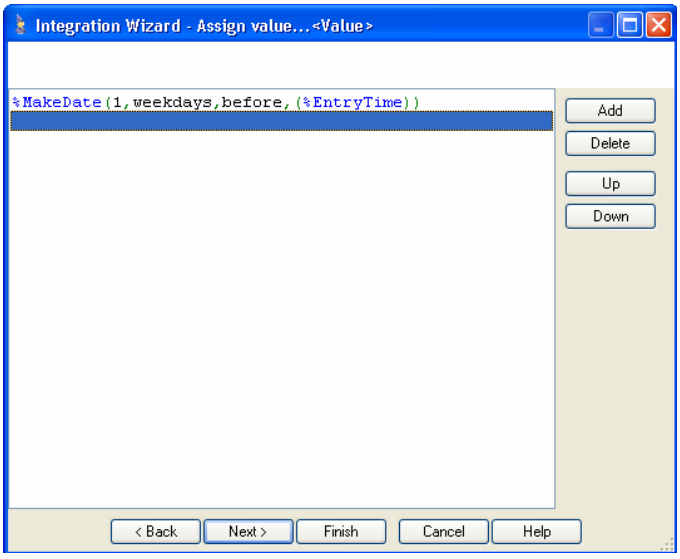


Figure 184: Integration Wizard - Value

10. Select **Finish** to close this instance of the Integration wizard.
    - The entry details are confirmed.

- The expression you just built is displayed in the **Value** field of the original instance of the Integration wizard.

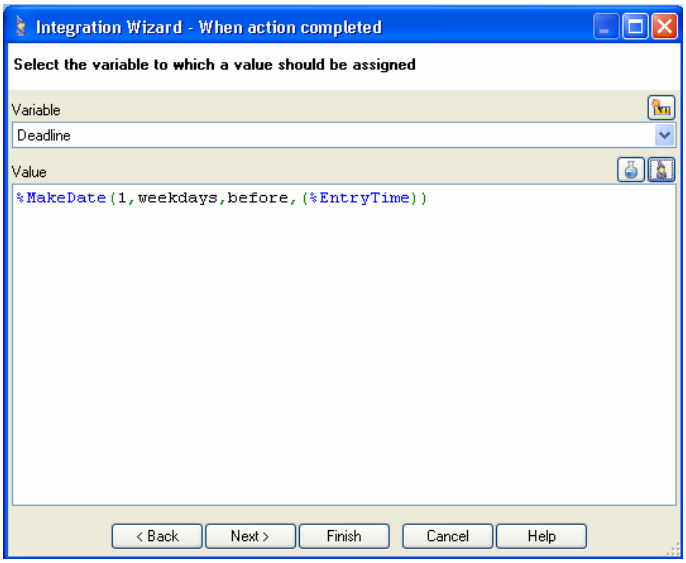


Figure 185: Confirm Assigned Value

11. Click on **Finish** to close the Integration wizard.
    - The final command is displayed in the *When action started* field of the action's **Do this** tab.
    - Place your cursor over the value to expand the display.

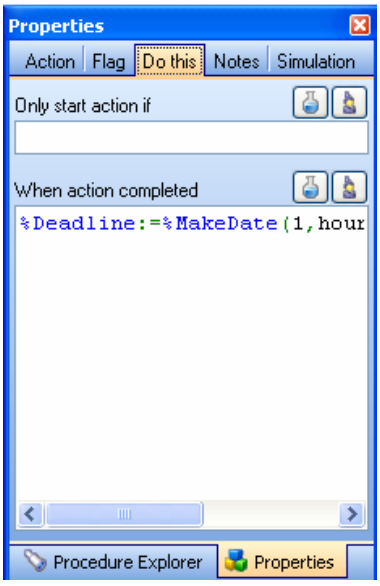


Figure 186: Formula displayed in When Action Started field

The use of multiple instances of the Integration wizard can be very useful in constructing complicated formulas.

 © Metastorm Inc., 2008

Integration Wizard Example: Conditional Operation

In this example, we will select the item “Conditional operation” and will use the Condition Builder to define the operation.

1. Select Action on the procedure map.
2. In the Properties editor select the Integration wizard.
3. Select the Conditional Operation and select the **Next** button.
4. In the Condition Builder field click on the icon next to the Formula Editor to open the Condition Builder, or the Formula Editor.
    - The Condition Builder; or
    - The Formula Editor is displayed.

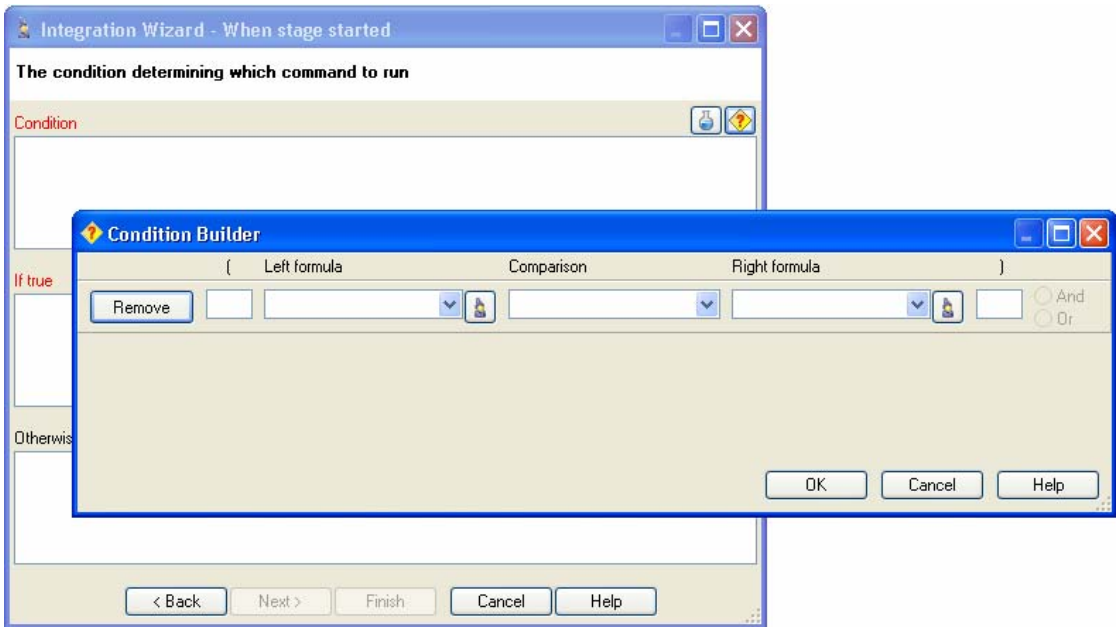


Figure 187: Conditional Operation Example

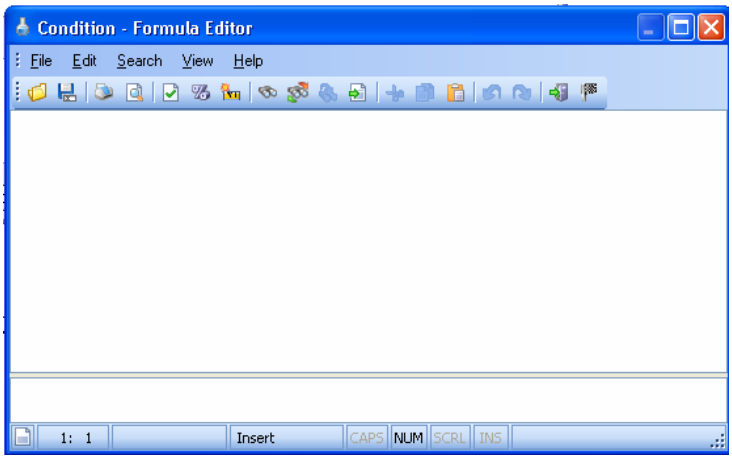


Figure 188: Formula Editor

5. The remaining formulas on this screen (*If true* and *Otherwise*) may also be created using either the Formula editor or the Integration wizard.

10.1.3 Condition Builder

There are a number of places within the Integration wizard where you can build up a condition. When required, Designer will display the Condition Builder dialog, see *Figure 187* above.

1. Enter a value manually, select a field whose value is to be tested from the list in the **Left formula** drop-down, or click on the **Integration wizard** button to build a formula for the value you require.
2. Select the comparison to be made from the **Comparison** drop-down.

    - You may select from the following options:

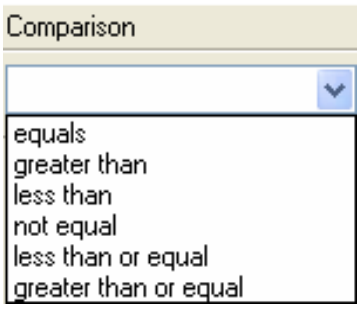


Figure 189: Comparison Options

3. Enter a value manually, select a field to be compared against from the **Right** formula drop down, or click on the **Integration wizard** button to build a formula for the value you require.
4. When you have completed the first line of the condition you can add more lines by clicking on the **And** button (if you want all your conditions to apply), or by clicking on the **Or** button (if you want any of your conditions to apply).
5. To remove a line from the condition click on the **Remove** button at the start of that line.
6. If you want to express an order of precedence by including parenthesis, enter them as appropriate in the left and right **Parenthesis ( )** fields.
7. Click on **OK** when you have finished building the condition.

    - The Integration Wizard is displayed with the entry in the Condition field.
    - If the condition is not valid, Designer will display a warning and you will need to correct it.

 © Metastorm Inc., 2008

10.1.4 Add-ins

At certain points within the Integration wizard, an ellipsis may be displayed. When you click on the ellipsis, an add-in dialog box is displayed for the entry of additional information:

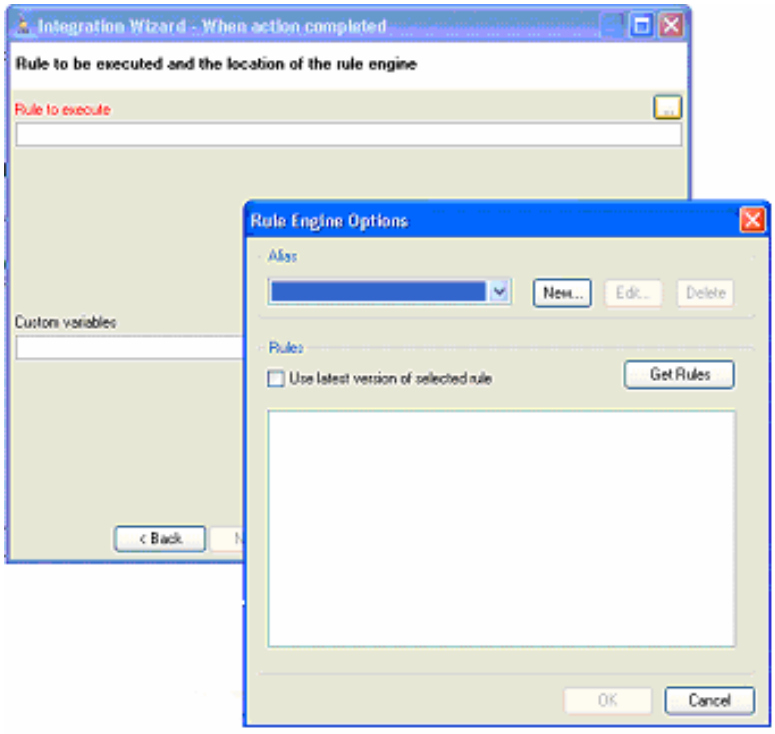


Figure 190: Add-in Dialog

1. Enter or select values for any fields in the add-in dialog.
2. Click on the **OK** button.

10.1.5 Formula Editor

In addition to the Integration wizard, you may also use the Formula editor to create or edit formulas of a property.

Figure 191: Formula Editor Button

This tool may be opened at any time by clicking on its button, see *Figure 188* above.

Using the Formula Editor

To use the Formula editor:

1. Once you have opened the Formula editor, you may type into or edit the existing content of the field. As you type, you will notice that the content of the field will be color-coded based on the type of field you are working in. The following table lists the field types and corresponding color-coding:

Field	Color	Background Color
IDENTIFIER	Default (Black)	Default (White)
STRING	Magenta	Default (White)
NUMBER	Red	Default (White)
SYMBOL	Green	Default (White)
LITERAL	Default (Black)	Default (White)
OPERATOR	Green	Default (White)
LOGIC_OPERATOR	Green	Default (White)
DELIMITER	Green	Default (White)
FUNCTION	Blue	Default (White)
KEYWORD	Default (Black), Bold	Default (White)
COMMENT	Dark Blue, Italic	Light Blue

Table 34: Formula Editor Color Codes

While you are editing a formula, clicking on the **Start Formula** button , or typing a percent sign (%). A scrolling list of valid entries is displayed. Selecting an entry and pressing **Enter** will insert that entry at the current cursor position.

2. When you are finished entering or editing the formula, select **Close** from the **File** menu or toolbar. If the formula has changed, you will be asked if you want to save your changes. You may also select **Save & Close** from the **File** menu or toolbar, which saves the formula and, if the formula passes evaluation, closes the Formula editor.

The Formula editor will check your formula and notify you in a lower pane if there are any errors. If it finds errors, the Formula editor will ask if you want to continue. In this case, select **No** to return to the Formula editor. Double clicking on the formula will select the line in the formula editor where the error was found.

3. You may also save the formula as you have typed it by selecting **Yes**. The error will be reported again if you run the verification function on the procedure, or try to publish it. You cannot successfully publish a procedure containing errors of this kind.

*If the content of the Formula Editor is part of a larger formula, but is not a formula itself, it will fail the evaluation. For example, when using the Integration Wizard item User Attribute, you enter the Where clause of an SQL query into the **Selection criteria** field. It is possible to edit the contents of this field using the Formula Editor, but the evaluation will always highlight errors because the Where clause is not a valid formula. In this case, click **Yes** to save the contents and continue.*

Samples of formula syntax may be found in the Formula Components *Appendix B.*

10.1.6 Accessing Databases

Designer uses ADO, an industry standard database interface, to access data in database management systems that use Structured Query Language (SQL) as a data access standard. Each

data provider has a data source name that stores information about how to connect to it. This section explains the arguments displayed by the Integration wizard when you are specifying access to a database.

Table Name(s) Argument

When specifying processing to be performed in a Metastorm or external database you may need to specify which table or tables are to be used. Enter the names of the database tables in the *Table name(s)* argument. You must specify the names of valid database tables. Alternatively, click on the Table browse button to the left of the argument so that you can select tables from the list provided.

When you click on the **Pick Table(s)** button the Pick Tables dialog is displayed:

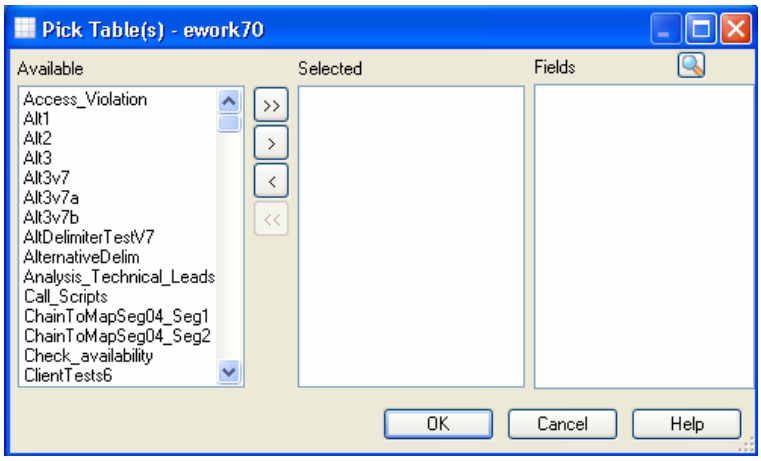


Figure 192: The Pick Tables dialog

1. Select the database from the **Database** drop-down.

   The list of available tables for that database in the **Available** list is updated.

2. To select a single table, click on it and then click on the > button. To select several tables, make the selection by clicking on the table names with the **Ctrl** button held down and then click on the > button. To select all tables click on the >> button.

   - The names of the selected tables are moved to the *Selected* list.

3. To remove tables from the **Selected** list use the < button to remove one or more tables that you have previously selected, or click on the << to remove all.
4. To see which fields are included in a table, click on the **Preview** fields button above the **Fields** list.
5. Click on the **OK** button when you have finished making your selection.

Column Name(s) Argument

When reading values from a database or folder you need to specify the columns from which to read data. Enter the names of the columns in the Column name(s) argument. You must specify the names of valid database columns. Alternatively, click on the drop-down list attached to the field so that you can select columns from the list provided.

Column Values Argument

When specifying processing that will update a database, using the Insert row item, you enter values for each column of a new row using the *Column values* argument. You can use the Integration wizard to assist you with this by clicking on the **Integration wizard** button or double clicking in the argument.

When inserting rows into a database in this way, you need to know, for each field, whether it holds text or numeric values. Text values should always be enclosed in 'single quotes'; numeric values should not.

Data Source Argument

When you use any of the functions that access an external database you need to specify the name of the data source in this argument. This is the name that identifies the data source. For example, the default Metastorm database has the Data Source Name (DSN) Metastorm. You can select a DSN by clicking on the drop-down *DSN* argument and selecting a defined DSN from the list. A new DSN can be set up using the ODBC Data Source Administrator.

User Name Argument

Many external databases require users to provide a database login name when making a connection. If you are using such a database enter your user name in the *user name* argument.

Password Argument

If you are using an external database that requires a password enter this in the *password* argument.

Row Condition(s) Argument

If you wish Metastorm BPM to select one or more rows in the database table, build a condition to specify which rows are to be included in the *Row condition(s)* argument. For example, you might enter *eUsername = '%User.Name'*, to affect only rows relevant to the user filling in a form. The SQL query builder is available to help you construct these clauses.

If you omit this, Metastorm BPM will select all rows in the selected table or tables. In the case of a dataset, Metastorm BPM ignores all but the first row that satisfies the specified criteria.

Be aware that selecting all rows in a database may have a significant adverse effect on performance.

10.1.7 Accessing Directories (Direct Directory Access)

Designer allows you to access data from a LDAP3 compliant directory (such as Novell eDirectory or Microsoft Active Directory) within a procedure. This section describes how to add an LDAP alias to a library and use it in a procedure.

Setting up the LDAP alias

In order to access data held in a directory, you must first set up an LDAP alias to the directory. The following steps are required to do this:

 © Metastorm Inc., 2008

1. Create a new library. In the Library Properties dialog, select the LDAP aliases tab.

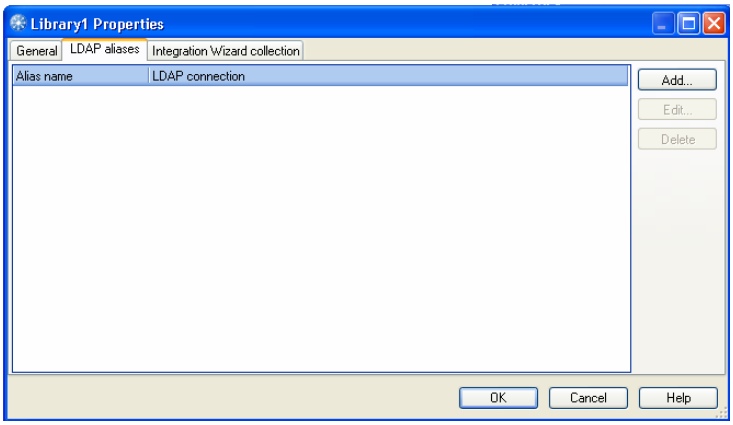


Figure 193: Library Properties LDAP aliases tab

2. Click on the **Add** button to add a new alias.
   - The LDAP dialog is displayed.

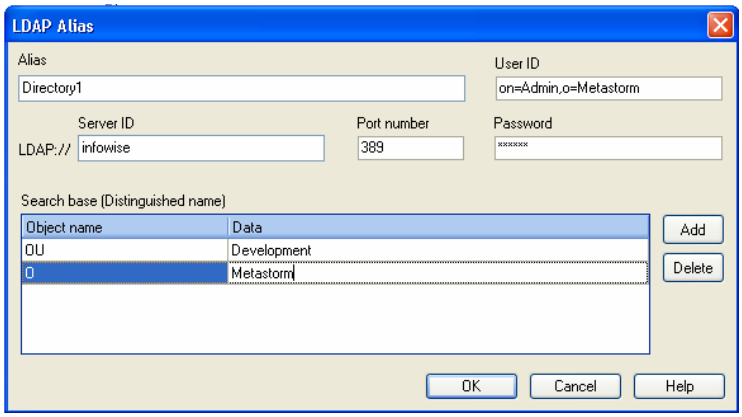


Figure 194: LDAP alias dialog

3. Enter the name of the alias and the LDAP connection details. The alias name may not be longer than 20 characters. Click on the **OK** button.

*The **Server ID** is restricted to the following characters: A-Z, a-z, 0-9.*

*The **Search base** is populated in reverse order, as shown in Figure 194, where the container "ou=Development" is listed before the root "o=Metastorm".*

4. Save, publish and close the library.

5. Open the procedure from which you want to access the directory data.

6. In the Procedure Properties dialog, select the Used Libraries tab.

7. Click on the **Add** button to add the library you have just published.
   - The library appears in the Used Libraries tab.

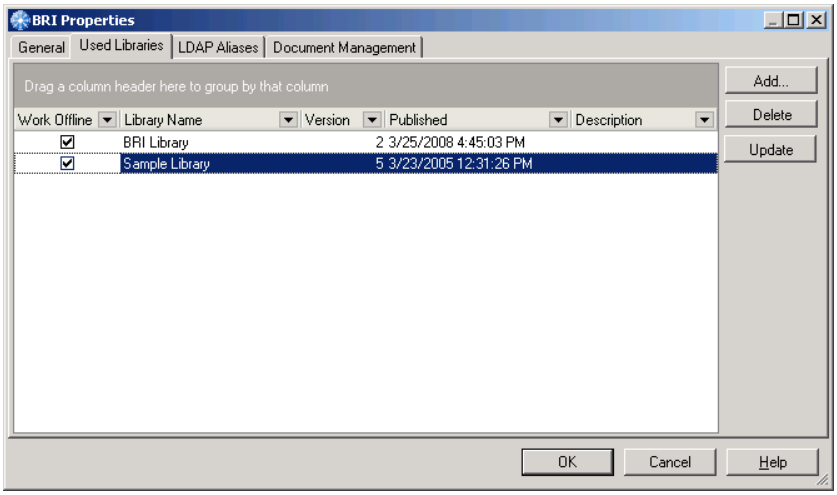


Figure 195: Procedure Properties Used Libraries tab

Searching the Directory

In the following example, we use the LDAP alias to retrieve details from the directory when a server operation button is clicked.

1. In the button's *When Button Pressed* property, select the Integration wizard icon.
2. From the **Action and Form Information** category, choose to **Assign Value**. Click on the **Next** button.
3. Select the Variable to assign the results to. In this case the custom variable, 'Subject'.
4. Click on the Integration wizard icon for the Value property.
    - The **Retrieve Information from LDAP directory search** setup screen is displayed.

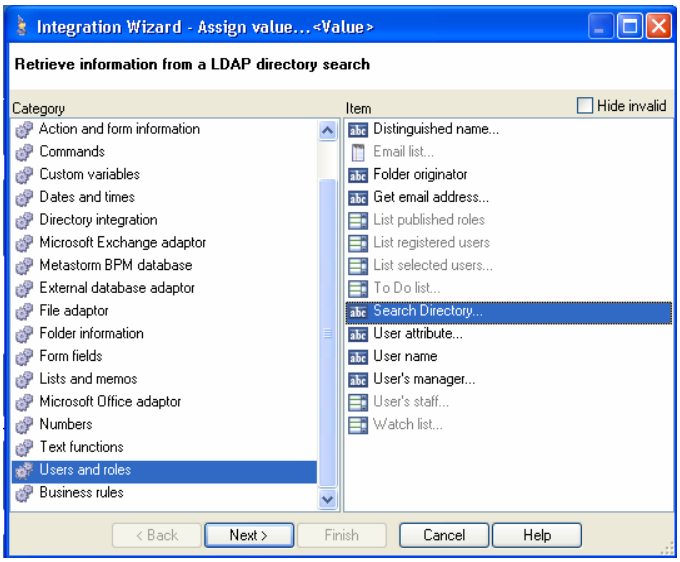


Figure 196: Integration wizard Users and Roles category

 © Metastorm Inc., 2008

5. From the **Users and Roles** category, choose the **Search Directory** item. Click on the **Next** button.

    - The attributes to be returned dialog is displayed.

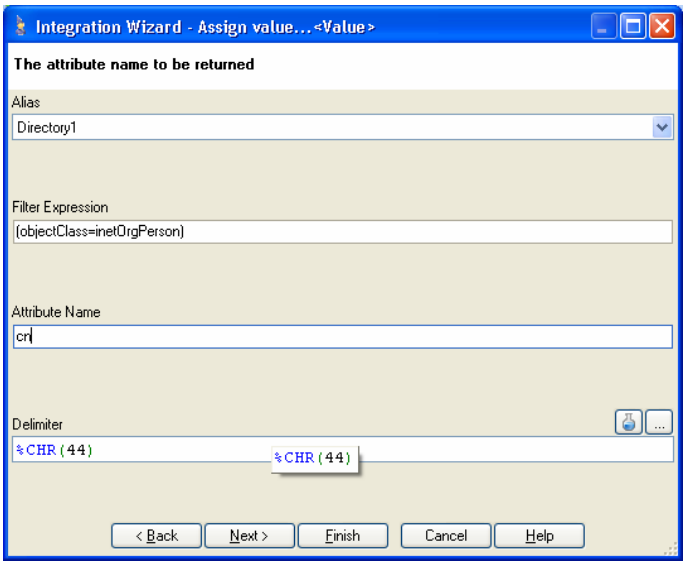


Figure 197: Search Directory parameters

6. Enter the following parameters:

    - **Alias**

    Enter the name of the LDAP alias that is mapped to the required directory.

    - **Filter Expression**

    Enter a filter expression to limit the results that are returned.

    The following are examples of valid filter expressions:

        - "(objectClass=inetOrgPerson)" returns all results where the object class is the inetOrgPerson.
        - "(&(objectClass=User)(OU=development))" returns all results where the object class is User and the OU is development.
        - "(|(objectClass=User)(OU=development))" returns all results where the object class is User OR the OU is development.

    - **Attribute Name**

    Enter the name of the directory's attribute that you want to retrieve. You may retrieve one attribute per %LDAPSearch function.

    - **Delimiter**

    Set the delimiter for the list to be returned, the default is a comma (chr44).

For further information on the parameters for %LDAPSearch, refer to the Formula Components Appendix.

7. Click on the **Finish** button. The formula is displayed in the Value property.

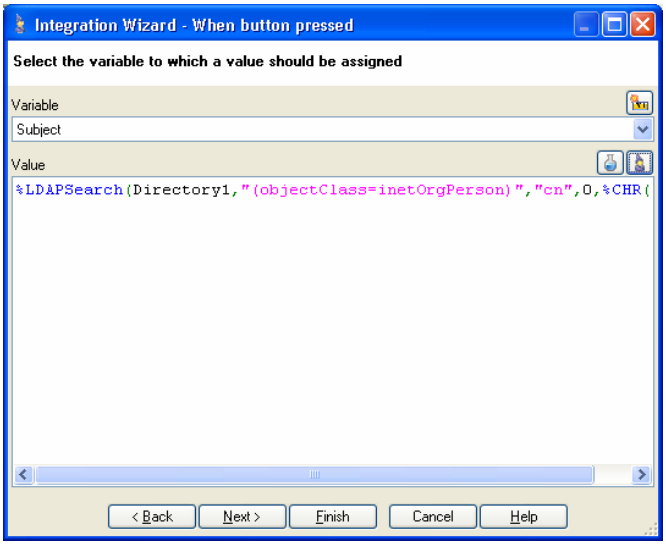


Figure 198: Setting Search Formula

8. Click on the **Finish** button.

    - The formula:

`%Subject:=%LDAPSearch(Directory1,"(objectClass=inetOrgPerson)","cn",%CHR(44))` has been built and appears in the Properties Editor of the button.

When the procedure has been published and the client user clicks on the Search Directory button, the information will be gathered from the directory and put into the Subjectvariable.

10.1.8 Custom Variables

Designer automatically establishes a number of variables that can be used in calculations or in reporting. In addition, when designing a procedure you can define your own custom variables and assign values to them.

Examples of variables that are defined and updated by Designer include:

- The name of a folder
- The name of the user that originated the folder
- The date and time at which a folder was created
- The date and time at which a folder was last updated

Custom variables are used either to store data that has been input by a user or extracted from an external system. Custom variables are associated with a particular map, and the data from the custom variable is stored in the map's database table. The custom variables form the columns of the table while the folders make up the rows.

When you select this category, Designer will display existing custom variables.

1. To create a new custom variable, click on the button.

    - Designer displays the **Custom Variables** dialog.

 © Metastorm Inc., 2008

2. Click on **Add**.

    - Designer displays the **Custom Variable** dialog:

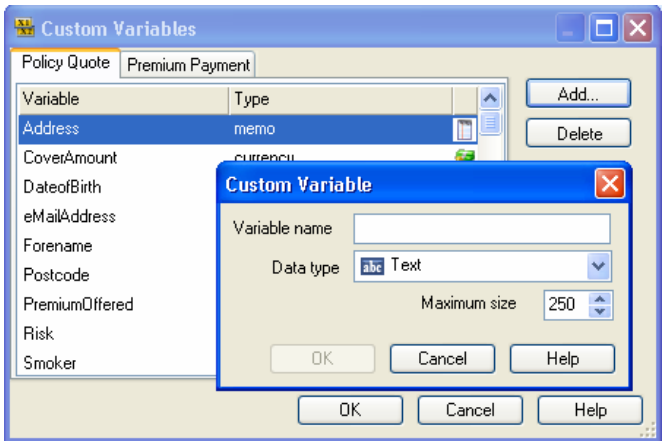


Figure 199: Custom Variables and Custom Variable dialog boxes

3. Enter a name for the variable in the **Variable name** field. This field may not be left blank, and must be unique for each new custom variable associated with this map. Variable names must be made of alphanumeric characters without spaces. No extended ASCII characters or non US English characters are allowed. If you are using SQL Server, the variable name may not be longer than 31 characters. If you are using Oracle, the variable name may not be longer than 30 characters.

4. Select the **Data type**. Designer automatically sets the default data type to match the field type you are defining.

    - Check
    - Currency
    - Date-time
    - Integer
    - Memo
    - Real
    - Text

5. If you are defining a *text* variable, you also need to enter a value in the **Maximum size** field, which is the size used to hold the name of any Metastorm object, such as a folder, server or user. The default size for text fields, radio group and drop-down fields is 250 characters. If you need to provide more room or allow for multiple lines of text, you should use a memo field and memo variable.

6. Click on **OK**.

    - The variable is added to the Custom Variables list:

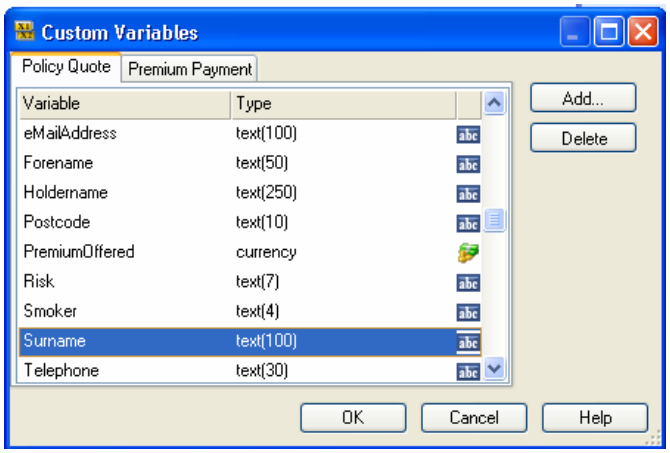


Figure 200: Sample Populated Custom Variables dialog

Custom variables must be associated with a specific map and are available to folders associated with that map; however, it is possible to use functions (such as flags and SQL queries) to make custom data from folders in one map available to folders in another map.

10.1.9 Defining External Tables

The External Tables feature lets you create supplementary tables from within the Designer. This eliminates the need to use database administration tools to create these tables externally on each machine running a specific procedure. External tables created using the Table Editor are stored as part of the database.

External tables may be used to simplify procedure creation. For example, if you want to populate a drop-down list with pick items, you can create the item list in a table rather than typing a list of items into the form element's property field.

Creating External Tables

To create an external table:

1. From the Menu Bar, click **View | External Tables**.
    - Designer opens the External Tables dialog.
    - If no tables have been defined, the dialog will display a message to that effect, similar to the following

 © Metastorm Inc., 2008

Figure 201: External Tables Dialog – No Tables Defined

2. From the **Table** options on the right, select the **Add** button.

    - Designer opens the *Add Table* dialog box.

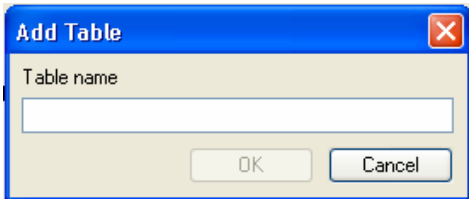


Figure 202: Add External Tables dialog

3. Type the name of the table you are creating and press **OK**.

    - The External Tables dialog is displayed with data entry facilities, see *Figure 201* above.

External Table names cannot include extended ASCII Characters, non US English characters or spaces and must not be Metastorm reserved words (words that have special meanings in the databases that Metastorm BPM support). Refer to the `Reserved Words.txt` *file for a list of reserved words. This file is located in the Metastorm BPM\Designer folder. In addition, they must not be system variable names. A list of system variables is contained within the* `oemdesigner.xml` *file under the <Variables></Variables> node. This file is located in the Metastorm BPM\Designer folder.*

4. To enter the information for the first column, click **Add** in the **Column** area of the screen.

    - The *Define Column* dialog box appears:

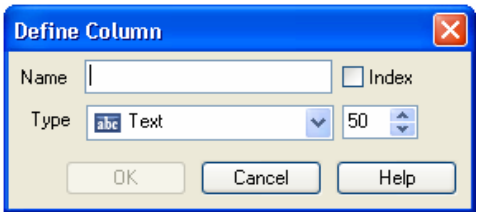


Figure 203: Define Column Dialog

5. In the **Column Name** field, enter the name of the column.

External Table column names cannot include non US English characters and must not be Metastorm reserved words (words that have special meanings in the databases that Metastorm BPM support). Refer to the `Reserved Words.txt` *file for a list of reserved words. This file is located in the Metastorm BPM\Designer folder.*

6. Select a column type from the **Type** drop-down menu. The choices are:

   - Check
   - Currency
   - Integer (whole number)
   - Real (number)
   - Date-Time (and/or date)
   - Text (single line, up to a maximum of 250 characters)
   - Memo (multiple lines, no effective size limit)

7. If you select **Text** as the column type, the property to the right becomes active. From this menu, select the column width in number of characters.

8. If this column is to be an index column for the database, check the **Index** checkbox.

9. Click **OK.**

10. Press **Enter** to create another column.

    Alternatively, select the **Add** button from the Column options on the right. You may edit column information by selecting the column and pressing the **Edit** button.

    After the procedure has been published, the columns can be edited in Metastorm BPM, but will not be updated in the database until the procedure has been re-published.

11. When you have finished entering column details, press **OK**.

When you save and publish the procedure, the table is created in the Metastorm database. It can then be used in the Integration wizard or Formula editor just like any other table stored in the Metastorm database.

10.1.10 Accessing Business Rules Engines

A procedure can invoke a rule from a Business Rules Engine, optionally passing system variables or custom variables to the rule. When the rule executes, it can return data to the procedure.

The general process for using Metastorm BPM with Business Rules Engines is:

1. A Business Rules Designer uses a Business Rules Composer tool to create rules.

   The Business Rules Administrator can incorporate system variables and custom variables into the rules via a .NET assembly Metastorm FolderData class.

 © Metastorm Inc., 2008

2. The Business Rules Administrator uses the Business Rules Engine to deploy the rules, so they can be accessed from the Designer and procedures.

   A web service, the Metastorm Rules Service, is hosted on the same machine as the Business Rules Engine.

3. The process designer uses the Designer to design a process. Where required, the process incorporates the `%ExecuteRule` function, as follows:

   a) The process designer publishes the BRI Library and references it in his procedure.

   b) The process designer opens the Integration Wizard.

   c) The process designer enters the authentication information for the Business Rules Engine.

   d) The Designer makes a call to the Metastorm Rules Service web service.

   e) The Metastorm Rules Service web service obtains a list of available rules, from the Business Rules Engine, and returns the list to Designer.

   f) Designer selects the required rule to be executed in the procedure, optionally specifying any custom variables to be passed to the Business Rules Engine.

4. The process designer publishes the procedure.

5. When the procedure is used, the Process Engine evaluates the `%ExecuteRule` function.

6. The Process Engine invokes the rule specified in the `%ExecuteRule` function.

The Metastorm Rules Service web service uses the Business Rules Engine, the .NET assembly Metastorm FolderData class and any specified system and custom variables to execute the rule.

7. If required, the Business Rules Engine uses the Rules Service web service to return a value to the Metastorm Engine.

The following diagram illustrates the process:

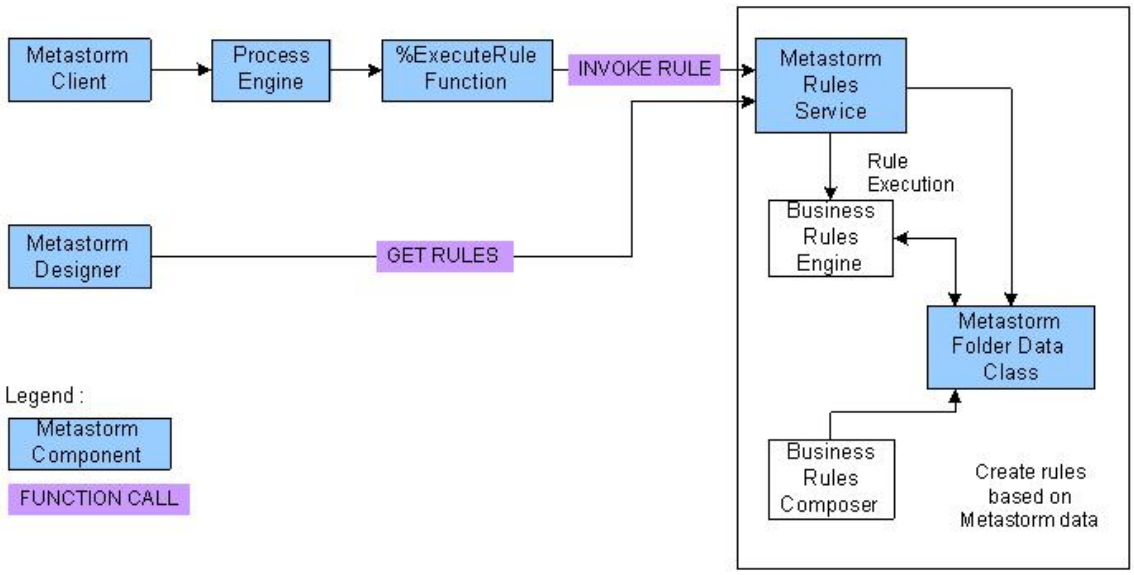


Figure 204: Use of Business Rules Engine with Metastorm BPM

The following subsections describe in more detail the use of the Designer to design a process.

Selecting the 'Execute a Rule' or 'Get Rule Result' Integration Wizard Item

1. Access the **Execute a Rule** or **Get Rule Result** Integration Wizard item, by:
   - starting an instance of the Integration Wizard, for example, from a **Rules** event, selecting the **Business rules** category, then selecting the **Execute a rule** item.
   - assigning a value to a variable by starting an instance of the Integration Wizard and selecting the **Business rules** category then selecting the **Get rules result** item (or from a calculation formula).

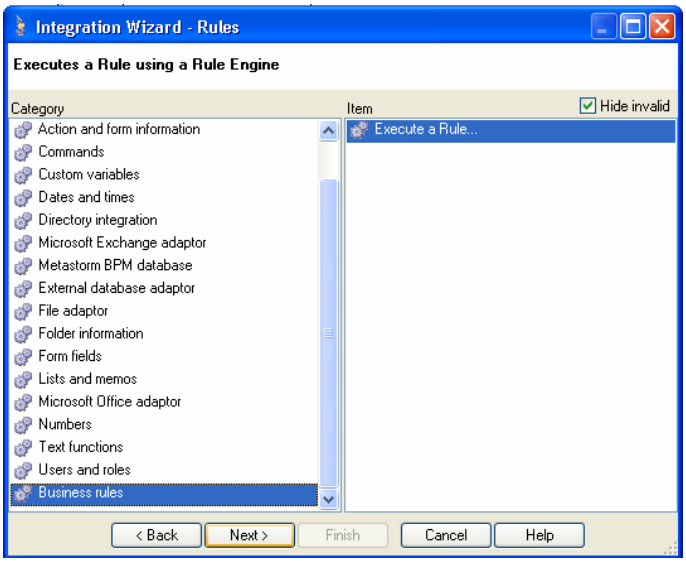


Figure 205: Business Rules Selection

2. Click the **Next** button.
   - The Rules Entry fields are displayed.

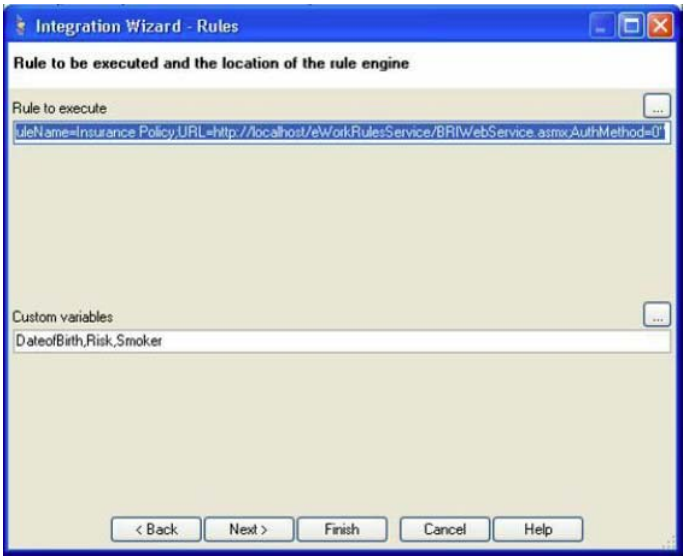


Figure 206: Business Rules Entry

 © Metastorm Inc., 2008

The Business Rules category and functions are stored in a collection in the BRI library which must be used in the procedure.

Specifying Rule Connection Information

1. Click on the ellipsis [...] next to the **Rule to execute** field.
    - The Rule Engine Options dialog is displayed:

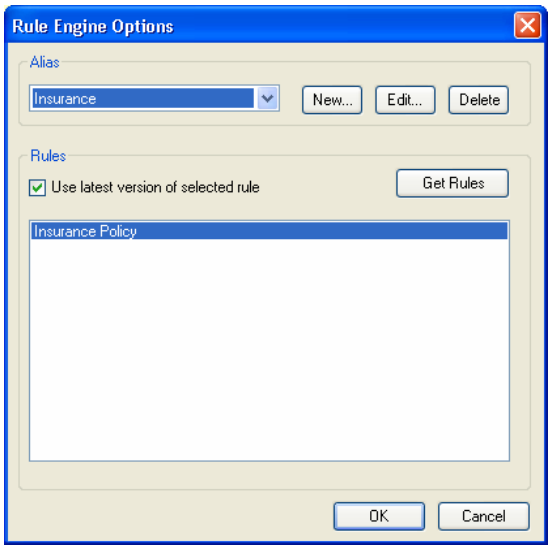


Figure 207: Rule Engine Options Dialog

2. Click on the **New** button. The Connection Details dialog is displayed:

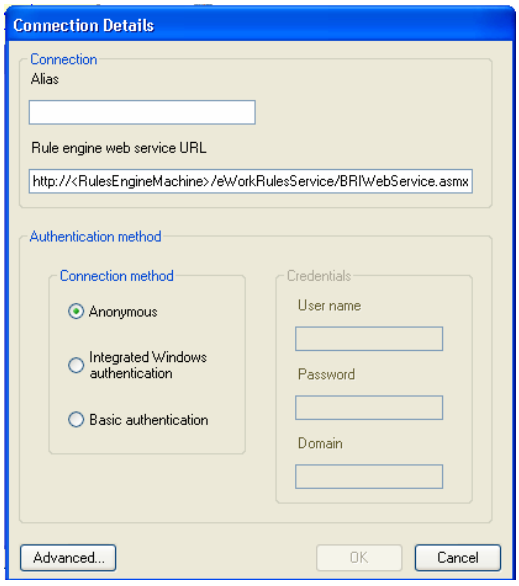


Figure 208: Connection Details

3. Enter the following details for the rule and its connection:

- **Alias** – an informative name for the rule. The **Alias** drop-down on the Rule Engine Options dialog contains all the aliases used on the machine.

  Rule Engine Web Service URL, e.g. for BizTalk 2006:

  http://<Machine Name>:81/BPMRulesService2006/BRIWebService.asmx

  Note, for BizTalk 2006, the port number for the URL depends on what port numbers are available when the installation is performed.

- **Connection Method**, as defined by an Administrator. We do not recommend using **Anonymous**. If you select **Basic Authentication**, we recommend using a secure (HTTPS) environment, as the authentication information is transmitted as clear text.

If the Metastorm Rules web service is on an IIS machine set up to use SSL, the machine calling the web service must have the certificate for the web service machine's web server in its trusted root certification authorities. It also needs a certificate from the certification authority.

- **Authentication Credentials**, if you selected **Basic authentication** as the **Connection Method**.
- Click on the **Advanced** button. The Advanced dialog is displayed:

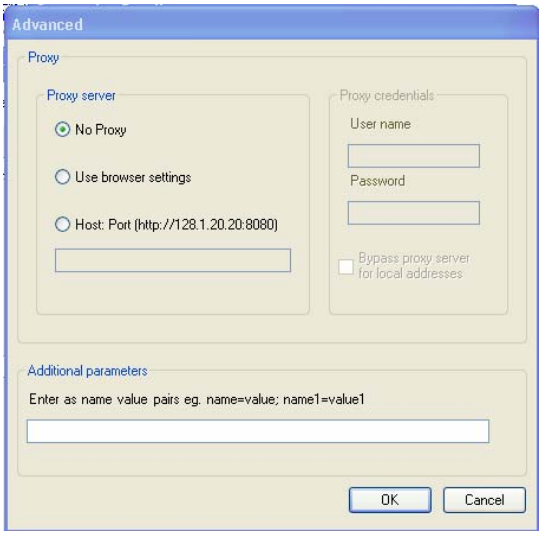


Figure 209: Advanced dialog

4. Enter the following Advanced details:
   - **Proxy server** details.
   - any **Additional Parameters** as name-value pairs, separated by semicolons.

For details of these parameters, refer to Appendix D - Web Service Configuration Parameters.

5. Click on the **OK** button on the Advanced dialog.
6. Click on the **OK** button on the Connection Details dialog. The **Alias** is now available in the drop-down list in the **Rule Engine Options** dialog.

 © Metastorm Inc., 2008

Selecting the Rule

1. In the Rules Engine Options dialog, see *Figure 207* above, click on the **Get Rules** button to connect to the business rules engine, using the connection information you have specified, and retrieve a list of available rules.
2. Select, from the list, the rule you want to associate with the selected **Alias**.
3. Check the **Use latest version of selected rule** checkbox, if you want the procedure always to use the latest available version of the rule, even without republishing the procedure.
4. Click on the **OK** button. The rule details are entered into the **Rule to Execute** field on the Integration Wizard.

Specifying Custom Variables to Pass to the Rule

1. Click on the ellipsis next to the **Custom variables** field. The Custom Variables dialog is displayed:

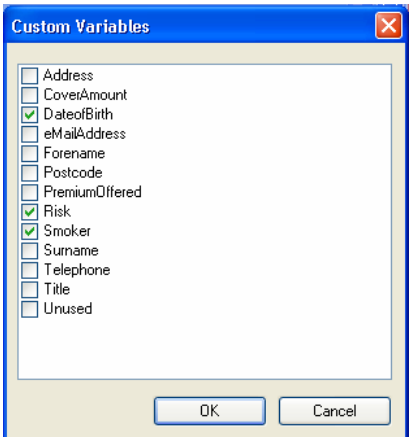


Figure 210: Custom Variables Dialog

2. Check the checkboxes for each custom variable you want to pass to the rule. If you don't select any custom variables, all custom variables are passed to the rule. All system variables are always available to the rule.
3. Click on the **OK** button. The custom variables are entered into the **Rule to Execute** field on the Integration Wizard.
4. Click on the **Finish** button to pass the rule, connection information and custom variables back from the Integration Wizard.

10.2 Scripting

Scripting allows process designers to implement new functionality into Metastorm BPM. Using scripting, process designers can add or extend Metastorm BPM functionality as follows:

- Access and modify folder and other variables (on the Process Engine machine).
- Make calls to functions (on the Process Engine machine).
- Access and modify field values (on the Metastorm Client machine).

- Integrate with third-party applications (on the Process Engine or Client machine).
- Write their own functions.

Within the Designer, the Script Editor, the Integration wizard, and the Formula editor may be used for scripting. These features allow you to:

- Write scripts. The Script Editor provides syntax highlighting for VBScript, JScript and JScript .NET. It also has links to VBScript, JScript and JScript .NET functionality. Alternatively, you can use an external text editor.

  To avoid errors occurring when scripts are run, it is a good idea to ensure that any variables within a script are declared within the script before they are used (and JScript .NET variables must be declared).

- With Jscript .NET, validate scripts.
- Attach a script to a form or map with the Scripts dialog.
- Call scripts from events using the Integration wizard or the Formula Editor.

If a script takes longer than 10 seconds to run, the Windows client user is given the option to stop it running. It is possible to delay or disable this option by setting the following registry key on the client's machine:

HKEY_CURRENT_USER\Software\Metastorm\ Client Components\ScriptTimeout

(ScriptTimeout is a string value).

To allow each script to run for more than 10 seconds, set ScriptTimeout to the required number of milliseconds.

To disable the option, set ScriptTimeout to -1.

If the registry setting does not exist, the option will be given after 10 seconds.

There are two types of scripts for Metastorm BPM:

- Server-side scripts are scripts executed on the machine running the Process Engine. These scripts may be attached to a map or map segment in the Scripts dialog or may be common to all maps in the procedure, and must be called using the functions %Script, %ScriptEval, and %ScriptAsync.
- Client-side scripts are scripts executed on the machine running the client. These scripts may be attached to a form or form segment in the Scripts dialog or may be common to all forms in the procedure, and are called in association with specific events through Client Extensions. Note that client-side scripts may only be written in VBScript or JScript.

Scripts may be common to a procedure or library or specific to a form, map, form segment or map segment. The following table outlines the availability of server and client-side scripts:

Procedure or Library	***Script Type***	***Associated with***	***Available to call from***
Procedure	Client	Common	Any form in the procedure
Procedure	Client	A Form	The specific form
Procedure	Client	A Form Segment	The form segment and any form the form segment is associated with
Procedure	Server	Common	Any map in the procedure
Procedure	Server	A Map	The specific map
Procedure	Server	A Map Segment	The map segment and any map the map segment is associated with

 © Metastorm Inc., 2008

Procedure or Library	*Script Type*	*Associated with*	*Available to call from*
Library	Client	Common	Any form in the library or the procedure in which the library is used
Library	Client	A Form	The specific form.
Library	Client	A Form Segment	The form segment and any form the form segment is associated with
Library	Server	Common	Any map in the library or the procedure in which the library is used
Library	Server	A Map Segment	The map segment and any map the map segment is associated with

Table 35: Script Availability

You can call either type of script via the Integration wizard or the Formula editor.

The use of non US English characters in scripting is not supported.

Scripts with duplicate FUNCTION names are not supported at publication as JScript and VBScript does not support overloading.

This section introduces the Metastorm *Scripts dialog* and illustrates methods of calling scripts; however, detailed information about script creation can be found in the *Metastorm Software Developer's Kit (SDK)*.

10.2.1 Working with the Scripts dialog

The Scripts dialog allows you to identify scripts that will be associated with the current procedure. Scripts included in the Scripts dialog may be called by their script name.

To open the Scripts dialog:

In the Designer, click on the **Scripts** button, or select **View** | **Scripts**.

The **Scripts** dialog appears:

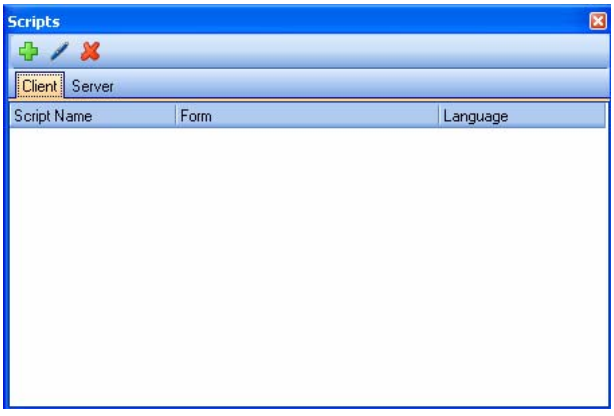


Figure 211: Scripts Dialog

The Scripts dialog can be docked on the top or bottom of the main pane or with Properties Editor.

Any scripts you have previously created are listed.

- To Add a script, click on the **Add** button .or press the **Insert** key.
- To Edit a previously created script, click on the **Edit** button .
- To Delete a previously created script, click on the **Delete** button .

If you delete or make changes to a previously created script, be sure you are aware how those changes will affect the Metastorm procedures that use the script.

Adding a Script

1. In the **Procedure Explorer**, select the map or form you wish to attach the script to.
2. Open the Scripts dialog, and click **Add**.

   The Script dialog appears.

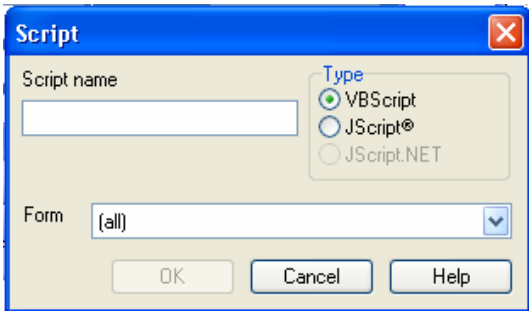


Figure 212: Script Dialog

3. In the **Script Name** field, enter the name of the new script.

   Script names cannot duplicate other script names in maps or forms (though script names can be duplicated under Common, provided that one is a procedure client script and one is a procedure server script or that a script with that name already exists in a used library).

   Script names can use characters which are configured with a default locale of North American, Western European or Central European character sets. However, script names cannot include any of the following characters: Pipe (|), Period (.) Dollar Sign ($), Equal Sign (=), Back Slash (\), Quotation Mark (“), Forward Slash (/), Percent (%), Comma (,), Plus (+), Colon (:), Semi-colon (;) and Angle brackets (<>).

4. In the **Type** list, select the scripting language type you are using—**VBScript, JScript** or **JScript .NET**. This will determine how the script text is syntax highlighted.

   JScript .NET is not supported for client-side scripts.

5. If you are creating a client side script, choose a **Form** or **Form Segment** from the dropdown. Select ‘(all)’ forms to create a common client side script. If you are creating a server side script, choose a **Map** or **Map Segment** from the dropdown. Select ‘(all)’ maps to create a common server side script.
6. Click **OK**.

 © Metastorm Inc., 2008

- The new script is added to the Scripts dialog.

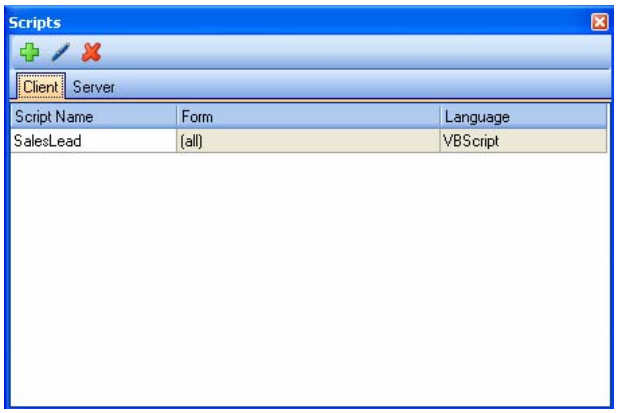


Figure 213: New Script Added

And the **Script Editor** dialog appears:

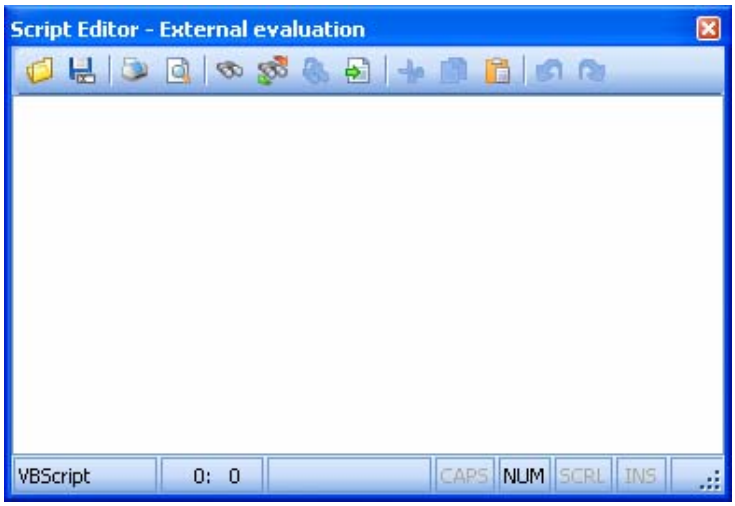


Figure 214: Script Editor

The Script Editor can be docked on the Scripts dialog. A tabbing facility is created to allow the use to toggle from one to the other.

7. You may now type a new script. As you type in the script, the code will be highlighted in different colors to differentiate between functions, values, and so forth. The syntax highlighting is dependent on the scripting language selected in Step 4 above.

   To change the font or color scheme of the text in the Script Editor, select **Options** from the **View** menu. On the **Editor** tab, choose the required font name and size. Note that not all fonts are supported.

8. When you have completed the script, go to the **Scripts** toolbar and click **Save.**

The **Scripts** dialog reappears. The script you just created has been added to the list.

The script is also added to the appropriate position on the Procedure Explorer. Common scripts are added to the **Common** item, and scripts associated with maps or forms are added as sub-items of the relevant maps or forms.

10.2.2 Calling Server-side Scripts

Server-side scripts are scripts executed on the machine running the Process Engine. These scripts may be attached to a map or map segment or they may be common to the procedure. You can attach server-side script calls to actions and stages. You can attach these scripts to form and field events, such as when an action or stage completes, when a form is loaded, or when a field value changes.

Calling a Server-side Script through the Integration wizard

You can use the Integration wizard to call server scripts with the following functions:

- **Run inline script** — Synchronous, returns no value. The Process Engine waits until the script completely executes before it continues that process.
- **Return script value —** Synchronous, returning a value.
- **Run async script —** Asynchronous, returns no value.

To access these commands via the Integration wizard, do the following:

1. Open the Properties editor for a selected item on either a map or a form.

   In this example, we will be defining properties for a Button, so the **Button Action** option *"Server operation"* has been selected on the **Button** tab of the Properties editor.

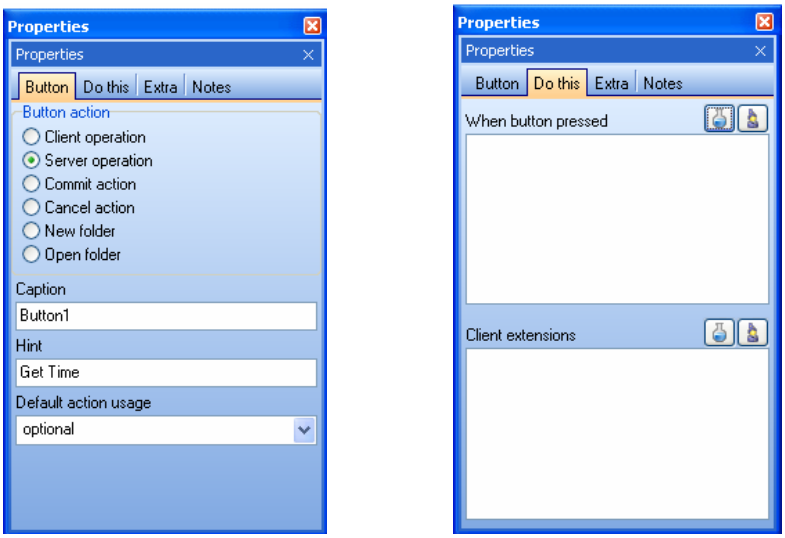


Figure 215: Properties editor Tabs

2. Place your cursor in the field in which you are calling a script.

   In this example the cursor has been placed in the **When button pressed** field on the **Do this** tab.

 © Metastorm Inc., 2008

3. Click on the **Integration wizard** button .

   - The Integration wizard opens.

4. From the **Category** menu, select **Commands.**

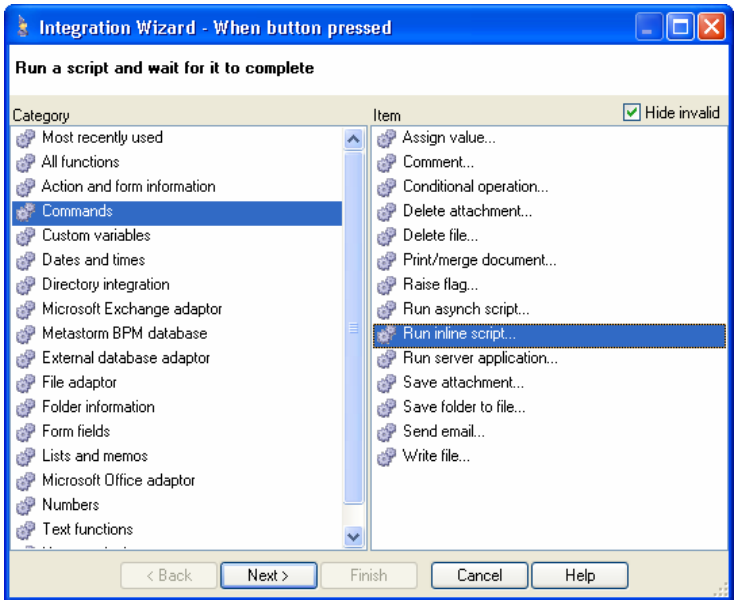


Figure 216: Integration wizard Main Menu

5. From the **Item** menu, select one of the functions shown in the following table, depending on how you wish to call the script:

Item To Select	***Function***
Run inline script	%Script
Return script value	%ScriptEval
Run async script	%ScriptAsync

Table 36: Functions and Corresponding Integration wizard Items

For this example, we selected "*Run inline script*" from the **Item** menu.

6. Click **Next**.

The following screen displays:

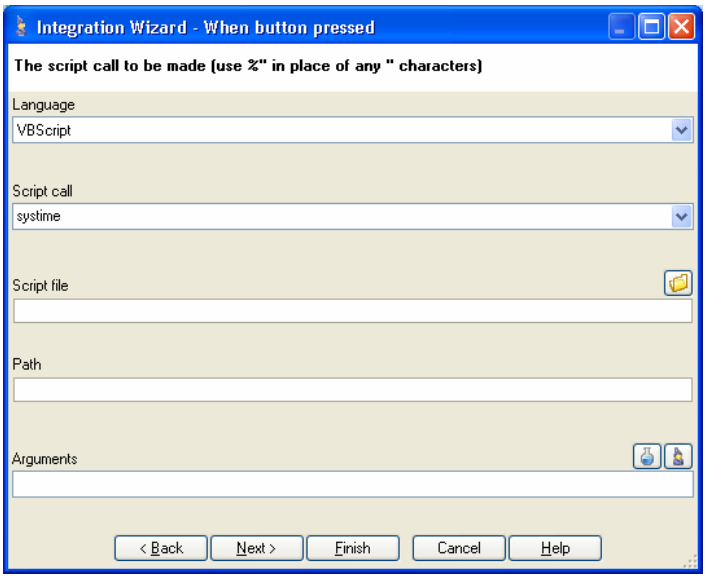


Figure 217: Integration wizard Script Call Dialog

7. From the **Language** drop-down menu, select the scripting language you are using—VBScript, JScript or JScript .NET.
8. In the **Script Call** field, type in the script call (which must be included in one of the scripts that has been previously added to the Scripts dialog), or the statement, which should include the function, plus any needed parameters. If any parameters require the use of quotation marks (**"**), as do string values in VBScript, use a percent sign (**%**) in place of each quotation mark.

In the example illustrated above, the script language selected was VBScript, and the Script call "systime" will be made to the script "System_Time," which was previously added to the Scripts dialog.

9. The **Script File** field contains the name of the original file containing global scripts. This file must be accessible from the server.

This step is optional and will rarely *be used. You enter data into this field only if you don't produce the script in the Metastorm Script Editor, but store it on the server as a separate file.*

You can fill in the **Script File** field in two ways:

- Type in the name of the original file containing global scripts. This file must be accessible from the server.
- Click on the **Browse** button located just above the **Script File** field. It opens a *File Open* dialog, which, if used, will fill in both the Script File field and Path field values.

10. In the **Path** field, type in the path to the specified file on the server. Note that this step, too, is optional and that you enter data into this field only if you don't produce the script in the Metastorm Script Editor, but store it on the server as a separate file.
11. If you are using JScript .NET, an **Arguments** field is also available. Use this to enter any number of required additional arguments.

 © Metastorm Inc., 2008

For further details, refer to the Scripting Developer Guide.

12. Click **Finish**.

When the procedure is run, the script will be called. In this example, when the button "Get Time" is pressed, a dialog will be displayed showing the current system time:

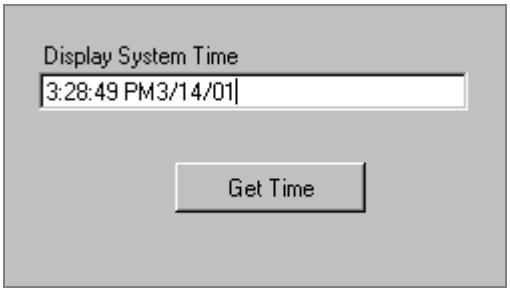


Figure 218: "Get Time" button script displaying system time

Calling a Server-side Script through the Formula Editor

You use the Formula editor to call server scripts with the following functions:

- **%Script** — Synchronous, returns no value. The Metastorm program waits until the script completely executes before it continues processing.
- **%ScriptEval —** Synchronous, returning a value.
- **%ScriptAsync —** Asynchronous, returns no value.

To access these commands via the Formula editor, do the following:

1. Open the Properties editor for a selected item on either a map or a form.

   In this example, we will be defining properties for a Button, so the **Button Action** option "Server operation" was selected on the **Button** tab of the Properties editor.

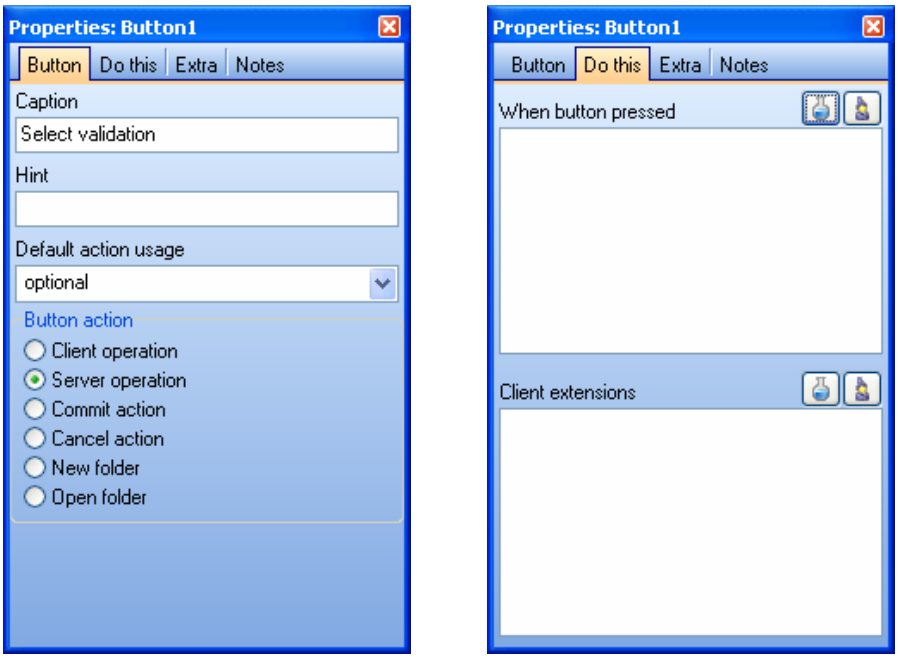


Figure 219: Properties editor

2. Place your cursor in the field in which you want to call a script. (In the sample screen above, the cursor has been placed in the **When Button Pressed** field.)

3. Click on the **Formula editor** button .

The Formula editor opens.

You can enter functions and variables in two ways:

- Type them in: or
- Select them from a menu. To see a menu of functions and variables, click on the **Start Formula** button (it has the percent [**%**] sign on it) in the toolbar (or just type a % character and wait for a few seconds). When the menu appears, click on the function or variable you need and it will automatically be placed onto the screen.

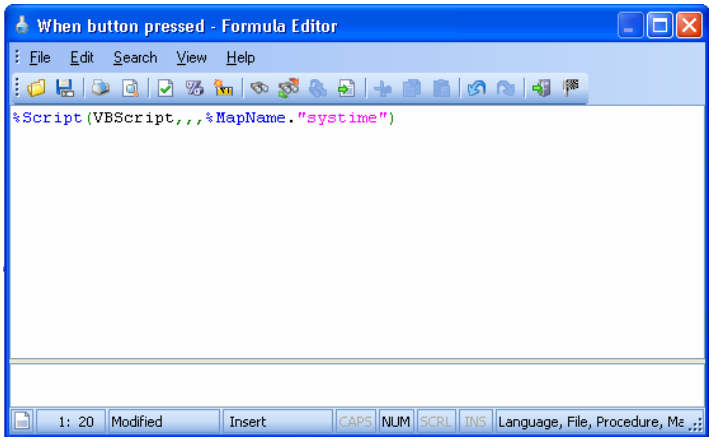


Figure 220: Formula Editor

 © Metastorm Inc., 2008

4. Type in the string needed to call the desired script.

   In the example shown above, the following script has been entered:

   ```
   %Script(VBScript,,,%MapName,"systime")
   ```

   The syntax contains the elements listed in the following table:

Syntax Element	*What It Contains*
%Script %ScriptEval %ScriptAsync	Function call which determines whether a file is synchronous or asynchronous and whether it returns a value
Language	VBScript, JScript or JScript .NET
Script File	Empty **unless** using a script stored in a file. Name of the file containing the script function you are calling on the server
Procedure Name	Empty
Map Name	Empty **if** it was a script in a file. Usually is %MapName. Must refer to the map containing the script function you are calling.
"Script Statement"	Usually a function call and any desired parameters in a language.
Arguments	JScript .NET only. Any number of additional arguments to be packaged up and passed into the entry point's `args` argument.

Table 37: Syntax Elements

If you need to input a quoted string as a parameter, you will need to place a % sign in front of the beginning and ending quote. See the above example.

5. When finished, click Save & Close .

When the procedure is run, the script will be called. In this example, when the button "Get Time" is pressed, a dialog will be displayed showing the current system time:

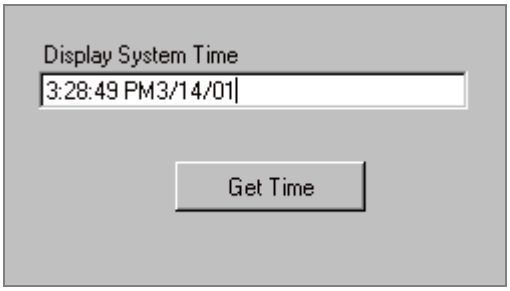


Figure 221: "Get Time" button script displaying system time

10.2.3 Calling Client-side Scripts

Client-side scripting allows Metastorm clients to interact with other applications. Client-side scripts may be attached to a form or form segment or they may be common to the procedure. Functions called in client-side scripts are synchronous, and do not return a value.

The following events are provided to trigger a script:

- Clicking a button

- Entering a field
- Exiting a field
- Loading a form.
- Submitting a form
- Canceling a form
- Entering a grid
- Exiting a grid
- Exiting the cell of a grid

You can use the Integration wizard or the Formula editor to call scripts to trigger these events.

Only one script call may be attached to each event.

Calling a Client-side Script through the Integration wizard

To attach a client script via the Integration wizard, do the following:

1. Open the Properties editor for a selected item on either a map or a form.

   In this example, we will be defining properties for a Button, so the **Button Action** option "Client operation" was selected on the **Button** tab of the Properties editor.

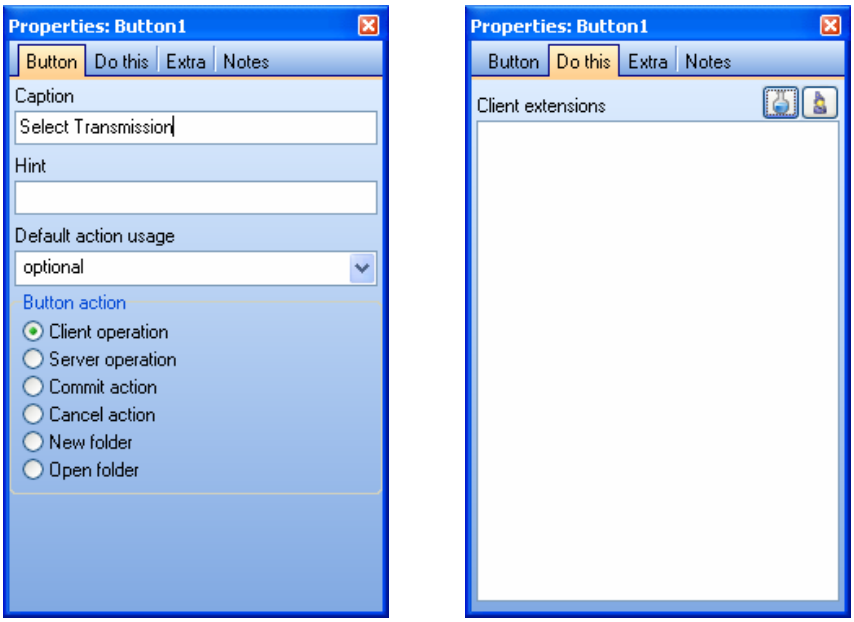


Figure 222: Properties editor

2. Click on the **Integration wizard** button .

   - The Integration wizard opens:

 © Metastorm Inc., 2008

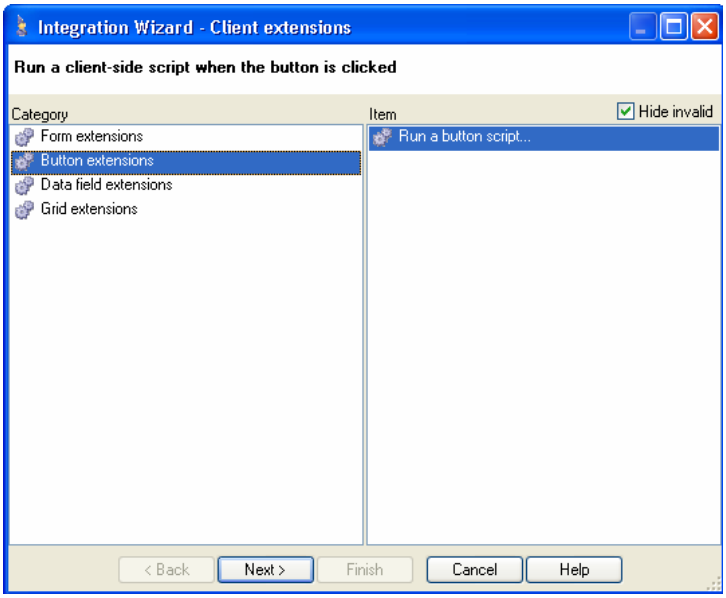


Figure 223: Integration wizard Main Menu

3. On the Integration wizard Main Menu you will see categories of functions or variables that can be used for calling a client-side script. To run a script to trigger one of the three client events, follow the instructions in the following table:

Client Event	*Category and Item to Select*
To run a script when a form is loaded:	From the **Category** menu, select **Form Extensions**. Next from the **Item** menu, select **Run a form script**.
To run a script when a form is submitted:	From the **Category** menu, select **Form Extensions**. Next from the **Item** menu, select **Run a submit script**.
To run a script when a form is cancelled:	From the **Category** menu, select **Form Extensions**. Next from the **Item** menu, select **Run a cancel script**.
To run a script when a button is pressed:	From the **Category** menu, select **Button extensions.** Next from the **Item** menu, select **Run a button script**.
To run a script when the user enters a field:	From the **Category** menu, select **Data Field extensions.** Next from the **Item** menu, select **Run a field entry script**.
To run a script when the user leaves a field:	From the **Category** menu, select **Data Field extensions.** Next from the **Item** menu, select **Run a field exit script**.
To run a script when the user leaves any cell of a grid.	From the **Category** menu, select **Data Field extensions.** Next from the **Item** menu, select **Run a cell exit script**.
To run a script when the user enters a *memo* field:	From the **Category** menu, select **Memo extensions.** Next from the **Item** menu, select **Run a field entry script**.
To run a script when the user leaves a *memo* field:	From the **Category** menu, select **Memo extensions.** Next from the **Item** menu, select **Run a field exit script**.

Table 38: Functions and Corresponding Integration wizard Items

*For details of fields on which the field exit and field entry events are available, refer to Table **40**: Availability of field exit and entry events.*

4. Select a category and item and press **Next** to continue.

    - For this example, the category **Button extensions** and the item **Run a button script** are selected.
    - The Client extensions screen is displayed:

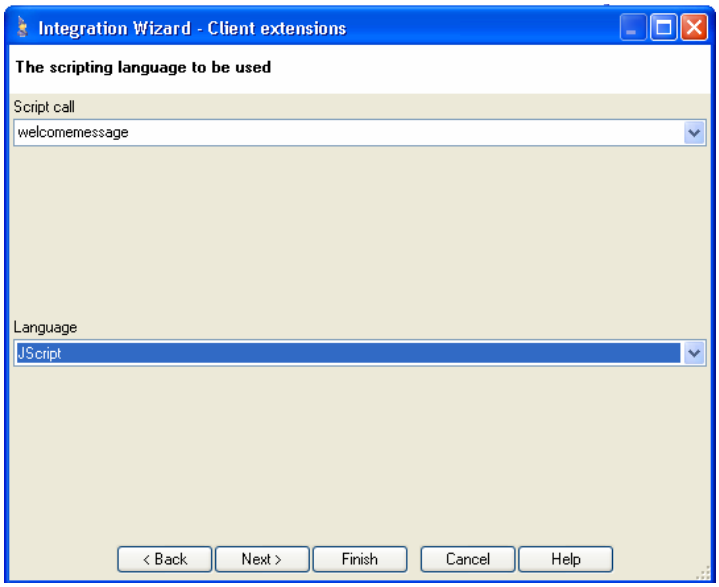


Figure 224: Integration wizard Script Call Dialog

5. In the **Script Call** field, type in the script call (which must be included in one of the scripts that has been previously added to the Scripts dialog), or the statement, which should include the function, plus any required parameters. If any parameters require the use of quotation marks (**"**), e.g. string values in VBScript, use a percent sign (**%**) in place of each quotation mark.
6. From the **Language** drop-down menu, select the scripting language you are using—VBScript or JScript.

In the example illustrated above, the script language selected was JScript, and the Script call "welcomemessage" will be made to the script "Welcome_Message," which was previously added to the Scripts dialog.

7. Click **Finish**.

    - The Client extensions field of the Properties editor will now contain the completed script:

 © Metastorm Inc., 2008

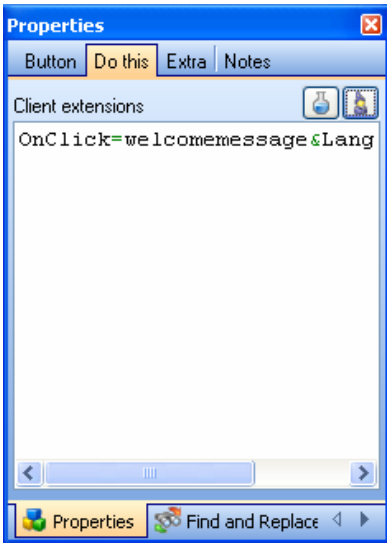


Figure 225: Properties editor – Client extensions field completed

When the procedure is run, the script will be called. In this example, when the button "Show Alert" is pressed, a dialog will be displayed showing a welcome message:

Figure 226: "Show Alert" button script displaying welcome message

Calling a Client-side Script through the Formula Editor

To call client scripts using the Formula editor:

1. Open the **Properties editor** for a selected item on either a map or a form.

In this example, we will be defining properties for a Button, so the **Button Action** option "Client operation" was selected on the **Button** tab of the Properties editor.

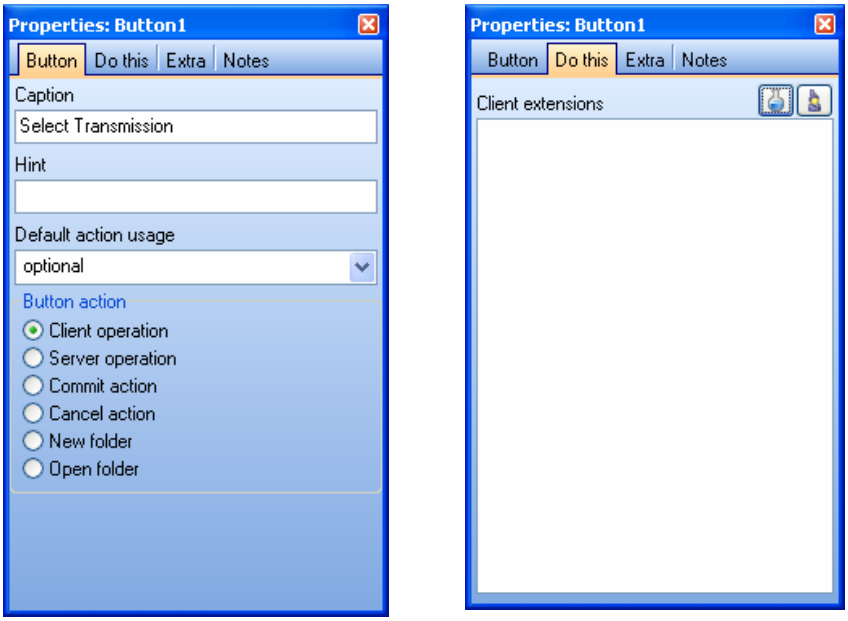


Figure 227: Properties editor

2. Click on the **Formula editor** button .
   - The Formula editor opens.
3. Type in the string needed to call the desired script.

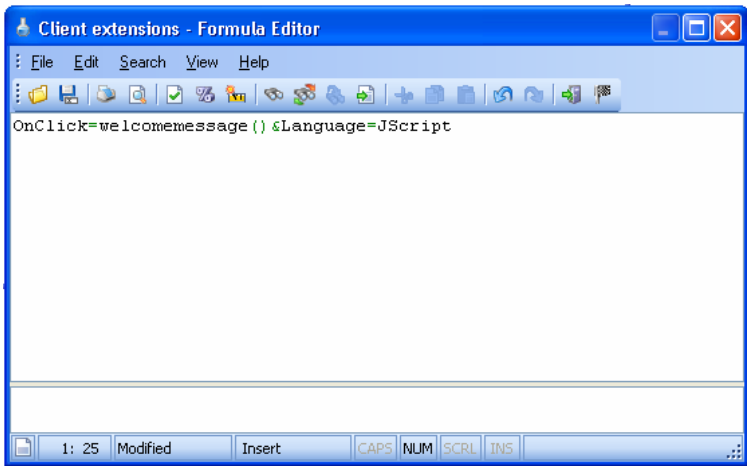


Figure 228: Entering a Client script in the Formula Editor

The following table shows the string needed for the five client events:

Client Event	Syntax String needed

 © Metastorm Inc., 2008

Client Event	*Syntax String needed*
To run a script when a form is loaded:	`OnLoad=<script function call>&language=<VBScript/Jscript>`
To run a script when a form is submitted:	`OnSubmit=<script function call>&language=<VBScript/JScript>`
To run a script when a form is cancelled:	`OnCancel=<script function call>&language=<VBScript/JScript>`
To run a script when a button is pressed:	`OnClick=<script function call>&language=<VBScript/JScript>`
To run a script when the user enters a field:	`OnFocus=<script function call>&language=<VBScript/JScript>`
To run a script when the user leaves a field:	`OnBlur=<script function call>&language=<VBScript/JScript>`
To run a script when the user leaves any cell of a grid:	`OnCellBlur=<script function call>&language=JScript`

Table 39: Client Events and Required Syntax String

*For details of fields on which the field exit and field entry events are available, refer to Table **40**: Availability of field exit and entry events.*

4. When finished, click **Save & Close** .

   - The **Client extensions** field of the **Properties editor** will now contain the completed script:

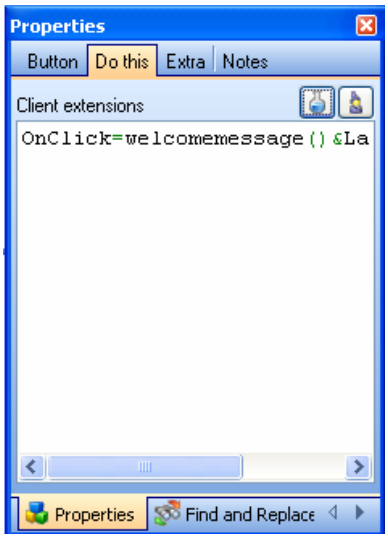


Figure 229: Properties editor – Client extensions field completed

When the procedure is run, the script will be called. In this example, when the button "Show Alert" is pressed, a dialog will be displayed showing a welcome message:

Figure 230: "Show Alert" button script displaying welcome message

Client Event Limitations

Field exit and entry

Scripts that cause message dialogs to be displayed or the focus to change are not supported when called by field exit or field entry events.

The field exit (OnBlur) and field entry (OnFocus) events are not supported on all fields. The following table illustrates the availability of these events:

Field Name	Field Exit Event Supported	Field Entry Event Supported	Note
Button	✕	✕	There is an OnClick event available
Check	✓	✓	
Clip	✕	✕	
Currency	✓	✓	
Date Time	✓	✓	
Dropdown	✓	✓	
Grid	✕	✕	There are field exit events (OnCellBlur) on each cell of an editable grid
Image	✕	✕	
Label	✕	✕	
List	✓	✓	
Memo	✓	✓	
Multiple Attachment Clip	✕	✕	
Number	✓	✓	
Radio	✓	✓	
Rule	✕	✕	
Signature	✓	✓	
Status	✓	✓	
Text	✓	✓	

*Table **40**: Availability of field exit and entry events*

 © Metastorm Inc., 2008

Client Side Script Functions

Information about client side script functions is provided in the Software Developer's Kit (SDK).

10.3 Glossary of Integration Wizard Items

The Integration wizard is divided into a number of categories, each of which has several predefined items that may be used when creating formulas in that category. Some items are specific to a category, while others may be used across multiple categories.

The following tables briefly describe the purpose of the different items available through the Integration wizard, grouped by category (individual categories are indicated by shaded bars). The first table lists Server Categories, while the second contains categories for use with Client Extensions. Suggestions for completing the arguments for each item are contained in the hints displayed when you use the Integration wizard.

Trailing ellipses in an item name indicate that the item requires arguments to be filled in by the user in a later Integration wizard screen.

SERVER CATEGORY	***ACTION***	***RESULT TYPE***	***DESCRIPTION***
Most recently used			Lists the most recently used items from all categories, as these are the ones you're most likely to want to use again.
All functions			Provides an alphabetized list of all items, in case you're not sure what category the item you want is listed under.
Action and form information			Lists items that are only relevant for an action or form. These are the ones you use most often in an action's "When action completed" properties, and in form and field properties.
	(text)…	text	Allows you to type in a literal text string. You can use this item to enter literal text when the Integration wizard expects a particular type of formula, or to by-pass the Integration wizard's type checking (which is what makes items disappear, or appear grayed if 'Hide invalid' is unchecked), by converting a non-text variable to its text representation. ***Note:*** *By using literal text, you force Metastorm BPM to ignore the formula type checking, and it is possible to enter an incompatible formula type. For example, you can put a memo variable into a text field; however, the memo variable may be truncated at run-time.*
	Action name	text	This returns the name of the current action. A value will only be available if an action has been invoked. You can use it when you want to display the current action name in a text field on a form or to include the current action name in an alert message.
	Assign value…		You use this option to assign a new value to a variable.
	Rule builder…	check	You use this option to open the Condition Builder to compare two values.

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Conditional operation…		You use this option to specify that Metastorm BPM is to perform one command if a condition is TRUE, and another if it is FALSE.
	Current date-time	Date-time	You use this option to return the date and time provided by the server on which the Process Engine is running.
	Current input	text	You use this option to specify that Metastorm BPM is to return the text entered by the user into the current field.
	Error message	Text	This is a "local" (to an action or form) variable, into which you can set any value (but typically, a description of why some user input is unacceptable) via the normal 'Assign value…' item – and then show in a (dependent) field on the form to give the user an explanation of what they should do.
	FALSE	check	You use this option to place a FALSE value in a check formula.
	Flag data…	text	You use this option to return any data associated with a flag. This is defined in the Flag Dialog. When you set up a flag in the Flag Dialog, you can specify data that will be passed when the flag is raised (flag data). This will appear in the flag dialog as a list of variables. Use the Flag data item to pass the value in one or more of these variables to the folder that is waiting for that flag to be raised. Enter the data in the Data line [1 to N] argument. For example, Action flag data[1] is the first variable in the list, Action flag data[2] is the second in the list, and so on.
	Flag folder	text	You use this option to return the folder ID of the folder (if any) that raised the flag that triggered this action. If a folder did not raise the flag, or an external application did not specify a folder when raising the flag, then the value will be blank.
	Flag name	text	You use this option to return the name of the flag (if any) that triggered this action.
	Form name	text	You use this option to specify that Metastorm BPM is to return the name of the form being used. If no form is being used then no value is returned.
	Grid selection…	text	You use this option to specify that Metastorm BPM is to return the value from the specified column when any row of a grid is selected.
	New note	memo	You use this option to specify the text to be saved as the note for an action.
	Next stage	text	You use this option to specify that Metastorm BPM is to return the name of the stage to which an action will move a folder.
	Server name	Text	You use this option to specify that Metastorm BPM is to return the name of the server interpreting the property formula.
	TRUE	check	You use this option to place a TRUE value in a check formula.
	User name	text	You use this option to specify that Metastorm BPM is to return the login ID of the active user. You might use this to define an alert message, or to populate a text field, with the name of the current user.

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
Commands			Most Metastorm formulas are declarative: they return a value, without producing any side effects. This category provides access to non-declarative formulas, all of which have some side effect, like updating the value of a variable, or outputting data to a file. Most of these items do not return a value. You're most likely to use these items in any "When <something happens>" property.
	Assign value…		You use this option to assign a new value to a Metastorm variable.
	Comment…		Adds comments to your code
	Conditional operation…		You use this option to specify that you want to perform one command if a condition is TRUE, and another if it is FALSE.
	Delete attachment…		Deletes an attachment from an attachment clip field in a folder
	Delete file…		You use this option to specify that Metastorm BPM is to delete a file on the server.
	List files …	memo	You use this option to specify that Metastorm BPM is to list the contents of a file directory on the server and place the list in a memo variable. This may be used to display the directory listing in a (calculated) memo field.
	Print/merge document…		When defining properties for an action you can specify that Metastorm BPM is to print a document containing data from the folder. If you wish to merge data from the folder into a Microsoft Word document you must set up a Word document with merge fields. The names of the merge fields in the Word document must be exactly the same as the names of the variables in the folder. If there are merge fields in the Word document that do not exist in the folder the print will fail. If there are fields in the folder that do not exist as merge fields in the Word document the print will succeed. Data to be merged with a Word document must not contain non-standard ASCII characters. The %Replace function can be used to ensure that non-standard ASCII characters such as " (ASCII 147) and " (ASCII 148) are not included in merge data. *Note: In Word 2003, when you create a document for mail merging, you have to tell the document that this is what it's being used for:* *1. From the menu select Tools | Letters and Mailings | Mail Merge...* *2. Do the first 2 of the 6 steps indicated by the mail merge window, typically using 'Letters' and 'Use the current document'.* *3. At step 3 close the mail merge window.* *4. Save your mail merge template document.*
	Raise flag…		You use this option to specify that a flag is to be raised. The use of this function allows one action to raise multiple flags.
	Read file…	memo	You use this option to specify that Metastorm BPM is to read the contents of a text file on the server and place the contents in a text or memo field. This may be used to display the files contents in a (calculated) memo field.

SERVER CATEGORY	***ACTION***	***RESULT TYPE***	***DESCRIPTION***
	Return script value…	text	Call a script and return a value
	Run async script…		Start a script on a separate thread
	Run inline script…		Run a script and wait for it to complete
	Save attachment…		Copy an attachment from a folder to a file
	Run server application…		You use this option to specify that Metastorm BPM is to run an application on the server where the Process Engine is running.
	Save folder to file…		Writes the values of all of the folder variables (Custom variables and Metastorm variables) to a comma separated values file. The first line of the file will consist of field names and the second will consist of values.
	Send email…		You use this option to specify that Metastorm BPM is to construct and send an email message. You may specify email addresses, or the names of any intended recipients. (Note: To use this option, the name of the recipients, must exist in the address book for the mail system in use. If you want the names to be created dynamically you can click on the **Integration wizard** button or type a formula in the field. You can also use the Formula editor to create a formula.) You can specify that the message be created dynamically by clicking on the **Integration wizard** button or type a formula in the field. You can also use the Formula editor to create a formula. You can also specify the address where reply messages will be sent if the mail system in use supports return addresses.
	Write file…		You use this option to specify that Metastorm BPM is to write text to a file on the server. The text can be the contents of Metastorm variables or the results of a formula. This can be either a new file, or you can append to an existing file.
Custom variables			Lists all custom variables defined for the currently selected map. Empty if no map is selected.
Dates and times			Lists all (non-custom) date/time variables, and functions that return or manipulate a date or time.
	Calculate date-time…	Date-time	You use this option to specify that Metastorm BPM is to calculate a new date-time a number of units before or after another date-time.
	Calculate duration…	Real number	You use this option to specify that Metastorm BPM is to calculate the interval between two date-times, in the specified units.
	Convert to date…	Date-time	You use this option to specify that Metastorm BPM is to attempt to convert a text value to a date.
	Convert to time…	Date-time	You use this option to specify that Metastorm BPM is to attempt to convert a text value to a time.

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Current date-time	Date-time	You use this option to return the date and time provided by the server on which the Process Engine is running.
	Folder deadline	Date-time	You use this option to specify that Metastorm BPM is to return the deadline set for this folder.
	Format date-time…	text	You use this option to format a date/time value for display.
	Read Metastorm date-time…	Date-time	You use this option to specify that Metastorm BPM is to read a date and/or time value from the database.
	Read external date-time…	Date-time	You use this option to specify that Metastorm BPM is to read a date and/or time value from a database.
	Read folder date-time…	Date-time	You use this option to specify that Metastorm BPM is to return a date and/ or time value from a folder in another map in this procedure.
	UNSPECIFIED DATE-TIME	Date-time	You use this option to specify that Metastorm BPM is to compare or reset time values to 'never'. ***Note:*** *The unspecified date used by Metastorm BPM is 1/1/1753 00:01. This will appear as a blank date in Metastorm BPM; however, reporting packages will report it as 1/1/1753.*
	When folder created	Date-time	You use this option to specify that Metastorm BPM is to return the date and time that this folder was created.
	When folder last updated	Date-time	You use this option to specify that Metastorm BPM is to return the date and time that this folder was last updated either when it entered the stage or was processed by a loop back action.
	When stage started	Date-time	You use this option to specify that Metastorm BPM is to return the date and time that this folder entered its current stage.
Directory integration			Lists all functions used with directory integration.
	Search Directory	text	You use this option to access data from a LDAP compliant directory.
Microsoft Exchange adaptor			Lists all functions used for emailing.
	Send email…		You use this option to specify that Metastorm BPM is to construct and send an email message. You may specify email addresses, or the names of any intended recipients. (Note: To use this option, the name of the recipient(s), must exist in the address book for the mail system in use. If you want the names to be created dynamically you can click on the **Integration wizard** button or type a formula in the field. You can also use the Formula editor to create a formula.) You can specify that the message be created dynamically by clicking on the **Integration wizard** button or type a formula in the field. You can also use the Formula editor to create a formula. You can also specify the address where reply messages will be sent if the mail system in use supports return addresses.
	E-mail list…	memo	Return an email address list corresponding to a list of Metastorm users.

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Get email address…	text	Return the email address of a specified user.
Metastorm database			Lists functions used to read and write information in the database.
	Datestamp format…	text	Format a date for lookup in any database using a SQL format.
	Delete attachment…		Delete an attachment from a folder
	Delete Metastorm rows…		You use this option to specify that Metastorm BPM is to delete rows from the database table.
	Execute procedure…		You use this option to specify that Metastorm BPM is to execute a database stored procedure.
	Execute stored procedure…		You use this option to specify that Metastorm BPM is to execute a database stored procedure. Arguments may be of any data type.
	Insert Metastorm row…		You use this option to specify that Metastorm BPM is to insert rows into the database table.
	List Metastorm data…	text	You use this option to specify that Metastorm BPM is to read a list of values from one column of the database and return them as a comma separated list of unique values. This is typically used to populate a drop-down field or other list in a form.
	New attachment…	text	Place a file from the server on an attachment clip field in a folder.
	Read Metastorm date-time…	Date-time	You use this option to specify that Metastorm BPM is to read a date and/or time value from the database.
	Read Metastorm number…	integer	You use this item to specify that Metastorm BPM should read a number from the database.
	Read Metastorm text…	text	You use this item to specify that Metastorm BPM should read a text value from the database.
	Read folder date-time…	Date-time	You use this option to specify that Metastorm BPM is to return a date and/ or time value from a folder in another map in this procedure.
	Read folder number…	number	You use this option to specify that Metastorm BPM is to return a number value from a folder in another map in this procedure.
	Read folder text…	text	You use this option to specify that Metastorm BPM is to return a text value from a folder in another map in this procedure.
	Save attachment…		Copy an attachment from an attachment clip field in a folder to a file
	Select Metastorm data…	text	You use this option to specify that Metastorm BPM is to select one or more rows from the Metastorm database.
	Timestamp format…	text	You use this option to specify an explicit text representation for a date-time in SQL format.

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Update Metastorm database…		You use this item to specify that Metastorm BPM is to update a database table in the database.
External database adaptor			Lists functions used to read and write information in other than Metastorm databases.
	Datestamp format…	text	You use this option to format a date/time value for display.
	Delete external rows…		You use this option to specify that Metastorm BPM is to delete rows from a database table.
	Insert external row…		You use this option to specify that Metastorm BPM is to insert rows into a database table.
	List external data…	text	You use this option to specify that Metastorm BPM is to read a list of values from one column of a database table and return them as a comma separated list. This is typically used to populate a drop-down field or other list in a form.
	Read external date-time…	Date-time	You use this option to specify that Metastorm BPM is to read a date and/or time value from a database.
	Read external number…	integer	You use this item to specify that Metastorm BPM should read a number from a database.
	Read external text…	text	You use this item to specify that Metastorm BPM should read a text value from a database.
	Select external data…	text	You use this option to specify that Metastorm BPM is to select one or more rows from a database.
	Timestamp format…	text	You use this option to specify an explicit text representation for a date-time in SQL.
	Update external database…		You use this item to specify that Metastorm BPM is to update a table in a database.
File adaptor			Lists functions used to read and write files.
	List files…	memo	You use this option to specify that Metastorm BPM is to list the contents of a file directory on the server and place the list in an memo variable. This may be used to display the directory listing in a (calculated) memo field.
	Read file…	memo	You use this option to specify that Metastorm BPM is to read the contents of a text file on the server and place the contents in a text or memo field. This may be used to display the files contents in a (calculated) memo field.
	Save attachment…		Copy an attachment from a folder to a file
	Save folder to file…		Writes the values of all of the folder variables (Custom variables and Metastorm variables) to a comma separated values file. The first line of the file will consist of field names and the second will consist of values.
	Write file…		You use this option to specify that Metastorm BPM is to write text to a file on the server. The text can be the contents of Metastorm variables or the results of a formula. This can either be a new file, or you can append to an existing file.
Folder information			Lists (non-custom) information held in the currently selected folder.

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	All notes	memo	Returns a formatted log of all the notes added to this folder via the %Action.Notes variable.
	Rule builder…	check	You use this option to open the Metastorm condition builder to compare two values.
	Earlier message…	text	You use this option to specify that Metastorm BPM is to return the alert message associated with a previous action on this folder.
	Earlier note…	memo	You use this option to specify that Metastorm BPM is to return the note associated with a previous action on this folder.
	Folder action count	integer	You use this option to specify that Metastorm BPM is to return the number of actions (including creation) that have been performed on this folder.
	Folder category	text	You use this option to specify that Metastorm BPM is to return the text value stored in the predefined Metastorm variable category.
	Folder deadline	Date-time	You use this option to specify that Metastorm BPM is to return the deadline set for this folder.
	Folder ID	text	You use this option to specify that Metastorm BPM is to return the internal identifier for this folder.
	Folder map	text	You use this option to specify that Metastorm BPM is to return the name of the map to which this folder belongs.
	Folder name	text	You use this option to specify that Metastorm BPM is to return the name shown on Watch and To Do lists for this folder. It will display the name of the folder, which is also what appears on the Watch and To Do lists, but not only that because it can return the name even if the folder doesn't appear on anyone's list at its current stage.
	Folder originator	text	You use this option to specify that Metastorm BPM is to return the name (logon ID) of the user who originally created this folder.
	Folder parent	text	You use this option to specify that Metastorm BPM is to return the folder ID of the folder (if any) of which this folder is a child folder. A folder will have a parent folder if it has been created in a Sub-procedure, or if it has been cloned from another folder, or if it has been created by a flag creation action and the flag has been raised by another Metastorm map. Its parent will be the folder that caused it to be created. Its parent could also be explicitly set by means of an Assign command.
	Folder priority	integer	You use this option to return the priority (0 to 9) set for this folder.
	Folder stage	text	You use this option to specify that Metastorm BPM is to return the name of the stage at which this folder now is.
	Folder subject	text	You use this option to specify that Metastorm BPM is to return the subject defined for this folder.
	Get child date-time…	Date-time	Returns a date-time value from a child folder
	Get child number…	integer	Returns a number value from a child folder
	Get child text…	text	Returns a text value from a child folder

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Procedure version	Integer	The version of the current procedure
	When folder created	Date-time	You use this option to specify that Metastorm BPM is to return the date and time that this folder was originally created.
	When folder last updated	Date-time	You use this option to specify that Metastorm BPM is to return the date and time that this folder was last updated. A folder is updated when it enters a new stage or as the result of a loopback action.
	When stage started	Date-time	You use this option to specify that Metastorm BPM is to return the date and time that this folder entered its current stage.
Form fields			Lists all data fields defined for the currently selected form. Empty if no form is selected or if no fields have yet been defined on the currently selected form. Data fields are shown in the format %field.<fieldname>.
Lists and memos			Lists all (non-custom) Metastorm memo variables, and functions used to return memos and lists.
	All notes	memo	Returns a formatted log of all notes added to this folder via the %Action.Notes field.
	COMMA	text	You use this option to insert commas into a list or reference a comma in the list.
	Convert list to memo…	memo	You use this option to specify that Metastorm BPM is to convert a comma-separated list to a (multi-line) memo.
	Convert memo to list…	list	You use this option to specify that Metastorm BPM is to convert a multi-line list to a comma-separated list.
	Count list entries…	integer	You use this option to return the number of entries in a comma separated list.
	Earlier note…	memo	You use this option to specify that Metastorm BPM is to return the note associated with a previous action on this folder.
	Find list entry…	integer	You use this option to return the position of an item in a list. For example, if Metastorm BPM were finding C in the list A, B, C, D, E, it would return 3.
	Line feed	text	You use this option to specify that Metastorm BPM is to insert a line feed (ASCII **<LF>**; decimal **<10>**) into a memos, or to search for a line feed in a memo.
	List Metastorm data…	text	You use this option to specify that Metastorm BPM is to read a list of values from one column of the database and return them as a comma separated unique list. This is typically used to populate a drop-down field or other list in a form.
	List external data…	text	You use this option to specify that Metastorm BPM is to read a list of values from one column of a database table and return them as a comma separated unique list. This is typically used to populate a drop-down field or other list in a form.

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	List Metastorm label/value pair…	text	You use this option to specify that Metastorm BPM is to read a list of values from two columns of the database and return them as a comma separated unique list. This is for use in populating a dropdown or grid dropdown column with label/value pairs. The label is shown in the field, and the value is passed to the database.
	List external label/value pair…	text	You use this option to specify that Metastorm BPM is to read a list of values from two columns of an external database and return them as a comma separated unique list. This is for use in populating a dropdown or grid dropdown column with label/value pairs. The label is shown in the field, and the value is passed to the database.
	List files…	memo	You use this option to specify that Metastorm BPM is to list the contents of a file directory on the server and place the list in an memo variable. This may be used to display the directory listing in a (calculated) memo field.
	List published roles	list	You use this option to specify that Metastorm BPM is to return the list of all roles published on the Metastorm server.
	List registered users	list	You use this option to specify that Metastorm BPM is to return a comma-separated list of all users (login Ids) registered to use Metastorm BPM.
	List selected users…	list	You use this option to specify that Metastorm BPM is to return a list of users with one or more specified attributes in common. The attributes will be taken from the Metastorm attributes table.
	NEW LINE	text	You use this option to specify that Metastorm BPM is to insert a new line into a memos, or to search for a new line in a memo.
	New note	memo	You use this option to specify the text to be saved as the note for this action.
	Read file…	memo	You use this option to specify that Metastorm BPM is to read the contents of a text file on the server and place the contents in a text or memo field. This may be used to display the files contents in a (calculated) memo field.
	Remove list entry…	List	You use this option to specify that Metastorm BPM is to remove a specified entry from a list. You need to specify the list from which the entry is to be removed and the position of the entry in the list (1, 2, 3 or whatever the position may be).
	To Do list…	List	You use this option to specify that Metastorm BPM is to return the list of users who have the specified folder on their To Do lists.
	User's staff…	List	You use this option to specify that Metastorm BPM is to return a list of the staff who have a specified role and report to the user you specify.
	Watch list…	List	You use this option to specify that Metastorm BPM is to return the list of users who have the specified folder on their Watch lists.
Microsoft Office adaptor			Lists all Microsoft Office related functions.

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Send email…		You use this option to specify that Metastorm BPM is to construct and send an email message. You may specify email addresses, or the names of any intended recipients. (Note: To use this option, the name of the recipient(s), must exist in the address book for the mail system in use. If you want the names to be created dynamically you can click on the **Integration wizard** button or type a formula in the field. You can also use the Formula editor to create a formula.) You can specify that the message be created dynamically by clicking on the **Integration wizard** button or type a formula in the field. You can also use the Formula editor to create a formula. You can also specify the address where reply messages will be sent if the mail system in use supports return addresses.
	Email list…	memo	Return an email address list corresponding to a list of Metastorm users. This will extract the email address from User details or the cached information from a directory if the Directory Integration Suite utilities are in use.
	Get email address	text	Get the email address of a specified user
	Print/merge document…		When defining properties for an action you can specify that Metastorm BPM is to print a document containing data from the folder If you wish to merge data from the Metastorm folder into a Microsoft Word document you must set up a Word document with merge fields. The names of the merge fields in the Word document must be exactly the same as the names of the variables in the Metastorm folder. If there are merge fields in the Word document that do not exist in the Metastorm folder the print will fail. If there are fields in the Metastorm folder that do not exist as merge fields in the Word document the print will succeed. The Word merge document must have no source file specified if it is to be used as a merge document. Data to be merged with a Word document must not contain non-standard ASCII characters. The %Replace function can be used to ensure that non-standard ASCII characters such as “ (ASCII 147) and ” (ASCII 148) are not included in merge data. *Note: In Word 2003, when you create a document for mail merging, you have to tell the document that this is what it's being used for:* 1. *From the menu select Tools | Letters and Mailings | Mail Merge...* 2. *Do the first 2 of the 6 steps indicated by the mail merge window, typically using 'Letters' and 'Use the current document'.* 3. *At step 3 close the mail merge window.* 4. *Save your mail merge template document.*
Numbers			Lists all (non-custom) Metastorm number variables, and functions used to return numbers.

SERVER CATEGORY	***ACTION***	***RESULT TYPE***	***DESCRIPTION***
	(currency)…	Currency	Allows you to type in a currency value
	(number)…	Integer	Allows you to type in a literal number
	Absolute number…	Integer	You use this option to return the absolute value of a number. This can be used to convert a negative number to a positive number. It has no effect on a positive number.
	Calculate number…	Integer	You use this option to specify that Metastorm BPM is to add, subtract, multiply or divide two numbers or expressions.
	Character code…	Integer	You use this option to return the ASCII code of a character.
	Conditional number…	Integer	You use this option to specify that Metastorm BPM is to return one number if a condition is TRUE, and another if it is FALSE.
	Count characters…	Integer	You use this option to return the number of characters in a piece of text.
	Find substring…	Integer	You use this option to specify that Metastorm BPM is to find where one string starts within another.
	Folder action count	Integer	You use this option to specify that Metastorm BPM is to return the number of actions (including creation) that have been performed on this folder.
	Folder priority	Integer	You use this option to return the priority (0 to 9) set for this folder.
	Format currency	Text	You use this option to specify an explicit text representation for a currency.
	Format number…	text	You use this option to specify an explicit text representation for a number. You may specify the number of decimal places that is to be displayed to and the minimum numbers of characters that the text representation should be. Additionally a number of formatting codes can be used. Valid format codes are: 0 pad with leading zeros (rather than spaces) - align to left (rather than right) within width + prefix positive numbers with explicit + sign p prefix positive numbers with a space character # always display decimal portion, even if zero b return empty string if value is zero %% append trailing % character For example: Formatting the number 36.2345 to 2 decimal places with a width of 10 characters and the format codes of + 0 would display +000036.23. Formatting the number 36.78 to 1 decimal places with a width of 6 characters and the format codes of # would display 36.8.
	Maximum number…	integer	You use this option to specify that Metastorm BPM is to return the largest of a list of numbers.
	Minimum number…	integer	You use this option to specify that Metastorm BPM is to return the smallest of a list of numbers.
	Read Metastorm number…	integer	You use this item to specify that Metastorm BPM should read a number from the Metastorm database.

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Read external number…	integer	You use this item to specify that Metastorm BPM should read a number from a database.
	Read folder number…	integer	You use this option to specify that Metastorm BPM is to return a number value from a folder in another map in this procedure.
	Remainder…	integer	You use this option to specify that Metastorm BPM is to calculate the remainder from the division of two numbers.
	Round number…	integer	You use this option to specify that Metastorm BPM is to round a number up or down to a specified number of decimal places.
Text functions			Lists Metastorm functions used to return (short) text values.
	(text)…	text	Allows you to type in a literal text string.
	Character…	text	You use this option to return the character value of an ASCII code.
	Character code…	integer	You use this option to return the ASCII code of a character.
	Comment…		Adds comments to your code
	Concatenate substrings…	text	You use this option to specify that Metastorm BPM is to concatenate two substrings. If you were to concatenate a variable containing the string *FirstName* with a variable containing the string *LastName*, the result would be *FirstName LastName*.
	Conditional text…	Text	You use this option to specify that Metastorm BPM is to return one text value if a condition is TRUE, and another if it is FALSE.
	Count characters…	Integer	You use this option to return the number of characters in a piece of text.
	EMPTY	Text	You use this option to specify that Metastorm BPM is to compare or reset a text, list or memo value to empty.
	Evaluate	Text	You use this option to evaluate a formula.
	Extract substring…	Text	You use this option to specify that Metastorm BPM is to extract a substring from a text or memo.
	Find substring…	Integer	You use this option to specify that Metastorm BPM is to find where one string starts within another.
	Get entry…	Text	Return a specified entry or line number from a list or memo
	Get email address	Text	Get the email address of a specified user
	Left…	Text	Return the left-hand part of a text string
	Read Metastorm text…	Text	You use this item to specify that Metastorm BPM should read a text value from the database.
	Read external text…	Text	You use this item to specify that Metastorm BPM should read a text value from a database.
	Read folder text…	Text	You use this option to specify that Metastorm BPM is to return a text value from a folder in another map in this procedure.

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Replace substrings…	Text	You use this option to specify that Metastorm BPM is to replace some or all occurrences of one string of characters within another.
	Right…	Text	Return the right-hand part of a text string
	SPACE	Text	You use this option to specify that Metastorm BPM is to compare or insert a single space into text.
Users and roles			Lists Metastorm variables and functions used to return user names and other related information.
	Distinguished name…	Text	Return a user's fully distinguished name (their unique ID within a directory)
	E-mail list…	Memo	Return an email address list corresponding to a list of Metastorm users.
	Folder originator	Text	You use this option to specify that Metastorm BPM is to return the name (logon ID) of the user (if any) who originally created this folder.
	Get email address…	Text	Return the email address of a specified user.
	List published roles	List	You use this option to specify that Metastorm BPM is to return the list of all roles published on the Metastorm server.
	List registered users	List	You use this option to specify that Metastorm BPM is to return a comma-separated list of all users (logon Ids) registered to use Metastorm BPM.
	List selected users…	list	You use this option to specify that Metastorm BPM is to return a list of users with one or more specified attributes in common. The attributes will be taken from the Metastorm attributes table. They can actually be taken from any attributes table that you specify in the next dialog.
	To do list…	list	You use this option to specify that Metastorm BPM is to return the list of users who have the specified folder on their To Do lists.
	Search Directory	text	You use this option to access data from a LDAP compliant directory.
	User attribute…	text	You use this option to find the value of a user attribute.
	User name	text	You use this option to specify that Metastorm BPM is to return the name, or logon ID, of the active user. You might use this to define an alert message, or to populate a text field, with the name of the current user.
	User's manager…	text	You use this option to specify that Metastorm BPM is to return the manager, who has a specified role, of the user you specify.
	User's staff…	list	You use this option to specify that Metastorm BPM is to return the staff, who have a specified role, of the user you specify.
	Watch list…	list	You use this option to specify that Metastorm BPM is to return the list of users who have the specified folder on their Watch lists.
MQ Messaging			Lists Metastorm functions used with message queuing software

 © Metastorm Inc., 2008

SERVER CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
	Send a message to a message queue		You use this function to send a message without having to use a pre defined queue.
	Send message to registered messaging queue		You use this function to send a message to a registered message queue.
	Add an attachment to existing Metastorm Folder		You can use this function to add a new attachment to an existing Metastorm folder. You can also use it to assign that folder to a clip field (in which case, you must assign the return value of the function to the clip field).
	Read an attachment from an Metastorm Folder		You can use this function to retrieve the Base64-encoded value of the requested attachment.
	Register an incoming queue definition		You can use this function to register a new in queue definition.
	Remove an incoming queue definition		You can use this function to remove an in queue definition.
	Register a new outgoing queue definition		You can use this function to register a new out queue definition.
	Remove an outgoing queue definition		You can use this function to remove an out queue definition.
Business Rules			Lists all Metastorm formulas that call Business Rules
	Execute a Rule		You use this option to execute a business rule on a Business Rules Engine, using the specified custom variables.
	Get Rule result		You use this option to execute a rule on a Business Rules Engine, using the specified custom variables, and return a result.
Event Handlers			Lists Metastorm External Event options.
	External Event Handler		You can use this function to create custom extensions. To use this feature the Process Events Library must be associated with a procedure. For further information refer to Appendix B, page 392 or Process Orchestrator for .NET Designer's Guide.pdf
	External Event Handler (boolean)		You can use this function to create custom extensions for conditional actions. To use this feature the Process Events Library must be associated with a procedure. For further information refer to Appendix B, page 392 or Process Orchestrator for .NET Designer's Guide.pdf

Table 41: Glossary of Integration wizard Server Categories

CLIENT CATEGORY	ACTION	RESULT TYPE	DESCRIPTION
Form extensions			Functions you can use in a form's client extensions
	Run a form script…		Runs a client-side script when the form is loaded.
	Run a submit script		Runs a client-side script when the form is submitted, allowing custom validation and custom error messages before any data is submitted. The script must return true for the submit to take place.
	Run a cancel script		Runs a client-side script when the form is cancelled. The form will always cancel.
Button extensions			Functions you can use in a button's client extensions
	Run a button script…		Runs a client-side script when the button is pressed.
Data field extensions			Functions you can use in a (non-memo) data field's client extensions
	Run a field entry script…		Runs a client-side script whenever the field receives the focus.
	Run a field exit script …		Runs a client-side script whenever the field loses the focus.
Grid extensions			Functions you can use in a grid field's client extensions
	Run a cell exit script…		Runs a client-side script whenever the cell of a grid loses the focus.
	Run a field entry script…		Runs a client-side script whenever the field receives the focus.
	Run a field exit script …		Runs a client-side script whenever the field loses the focus.
ComplexTypes SupportLibrary			Functions you can use with complex types.
	FormatComplex TypeDirect		Returns a formatted string using a complex type.
	FormatComplex Type		Returns a formatted string using a complex type.
	GetListDirect		Returns a string using a complex type array or collection.
	GetList		Returns a string using a complex type array or collection.
	GetNameValue ListDirect		Returns a name and value string using a complex type.
	GetNameValue List		Returns a name and value string using a complex type.

Table 42: Glossary of Integration wizard Client Categories

 © Metastorm Inc., 2008

Appendix A - Sample Procedures

A set of sample procedures is supplied with Metastorm BPM. They are intended to help you to understand some of the features available in the product. Each of the procedures illustrates a business process that may be built with Metastorm BPM.

You can use these procedures as a starting point for your own process development, or modify them for your own use.

Table 43 below lists the sample procedures that are shipped with Metastorm BPM. A sample library used by these procedures is also provided. In addition, the Send Mail procedure requires the Mail library.

Name	***Description***	***Uses Sample Library?***
BRI	Demonstrates the use of Metastorm BPM with a business rules engine. In addition to the Sample Library, this requires the BRI library, which is located in: `<installfolder>\Metastorm BPM\Designer\Sample Procedures\BRI 2006`	Yes
Case Management	A procedure with no predefined route. Each time a user takes an action, they decide who needs to act on the folder next.	Yes
External PO	Demonstrates the integration with BizTalk Servers offered by the Metastorm BPM Accelerator for BizTalk.	No
Flight	Used to illustrate a travel authorization and booking process.	Yes
Lead Tracking	Used to collect sales leads and track their progress through the sales cycle.	Yes
New Hire	Used to ensure that the necessary administrative processes are carried out when a new employee joins the organization.	No
News	A map for maintaining a discussion database, comprising a list of topics and associated note threads, along with an Administrative Form for browsing available topics by category.	-
One Call	Used by a call center to process and track customers calls.	Yes
Purchase Order	Demonstrates the use of an editable grid to capture line detail information in a purchase order.	Yes
Send Mail	Demonstrates the sending of mail to an SMTP server or a folder	Uses the Mail.xel library
Set Up	Used to add sample users and set up external tables. Helpful in setting up other sample procedures.	No
Training	Used to establish new training classes and book students in to the classes.	Yes
Wizard	Demonstrates the use of chained actions to provide a wizard-style interface.	No

Table 43: Sample Procedures List

Publishing a Sample Procedure

Locating the sample procedures

By default, the files needed to publish the sample procedures are installed in:

C:\Program Files\Metastorm BPM\Designer\Sample Procedures.

Using the sample library

A number of the sample procedures make use of a sample library, Sample Library.xel. Before publishing these procedures, the sample library must be published. By default, the sample library is also installed to C:\Program Files\Metastorm BPM\Designer\Sample Procedures.

For further information about libraries, refer to Section 2 of this document

Setting roles for the sample procedures

Use the Users and Roles tool to establish the users for the procedure and associate them with any roles that the procedure may use. If you wish, you may add yourself as the only user, giving yourself the 'Administrator' role, and use the Set Up sample procedure to add additional users.

To evaluate the procedure on a single machine in a testing environment, you may find it convenient to temporarily change all of the 'To Do List' roles in the procedure to 'everybody'. Alternatively, use the Users and Roles tool to temporarily assign all roles to yourself.

Using the Set Up sample procedure

The Set Up procedure can be used to prepare the sample procedures for use. It sets up users and role information and external table data. See below for further information.

The remainder of this Appendix contains notes on the sample library, and on each of the sample procedures.

Sample Library

This library contains the following sample forms, used by other sample procedures:

- The 'Notes' form displays, in a memo field, a historical summary of all notes entered against a folder.
- The 'Make a Note' form allows a new note to be entered against a folder.
- The 'Audit Trail' form displays a grid of all actions taken against a folder, in the order in which the actions were taken.

Points to Note

- The 'Make a Note' form uses the standard `%Action.Notes` variable to capture the user's input.
- The 'Notes' form uses the standard Integration Wizard All Notes item to present a formatted, time-ordered list of all notes entered against a folder (including, but not necessarily limited to, notes entered via the 'Make a Note' form).

 © Metastorm Inc., 2008

Set Up

This procedure is provided to ease the process of setting up the users and roles information and external table data for the more complex Metastorm sample procedures.

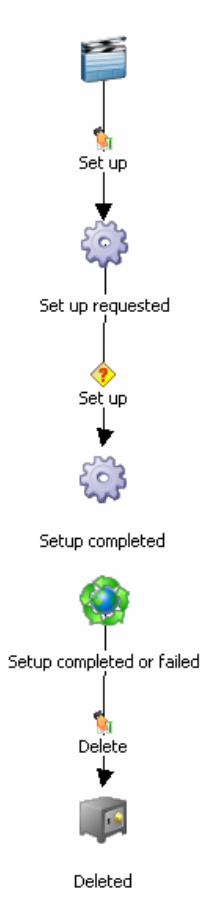


Figure 231: Set Up Sample Procedure

The procedure automatically registers 12 sample users (without Metastorm passwords):

Bertie Blauer	Jules Joiner	Nick Nurse
Cathy Cartright	Klaus Krammer	Paul Pascal
Claudia Clueup	Lara Linkup	Sam Smith
Jon Jackson	Lucy Laughalot	Sarah Singalong

It also:

- Inserts data into the Metastorm Attributes table. This table is used to define the roles and teams for both the Case Management and the One Call procedure.
- Builds a staff hierarchy for use by the 'Metastorm Manager' functions in the Flight procedure.
- Inserts initial data into the Flight_Airport table that is used to populate the 'Airport' drop-down fields in the Flight procedure. The publication of this procedure creates the Flight_Airport table in the Metastorm database.
- Inserts initial data into the Purchase_Order_Item_Code table that is used to populate the 'Item' drop-down column in the Purchase Order procedure. The publication of this procedure creates the Purchase_Order_Item_Code table in the Metastorm database.

Preparation

No preparation is required other than publishing the procedure. Only a user with the 'Administrator' role may run the procedure.

Points to Note

- The conditional 'Set up' action (from the 'Set up requested' stage) causes the (possibly lengthy) database operations to be performed by the Process Engine as a background activity, without forcing the user who submitted the original request to wait for these operations to complete.
- The 'ExecSQL' on the initial request action removes the entry for this procedure from users' Blank Forms, to prevent any attempt to run the set up procedure multiple times.
- The 'Setup completed or failed' common stage allows the 'Delete' action (which removes the users, attributes and additional table entries created by the original request) to be called whether or not the original set up succeeded.

 © Metastorm Inc., 2008

Case Management

This process demonstrates how Metastorm BPM can be used for a case management solution. It is impossible to define in advance the route the folder will take, so it is left to the user to direct the folder to the next person. When a task is defined, the person defining the task indicates both the team whose responsibility it is and the person in the team who will initially review the task.

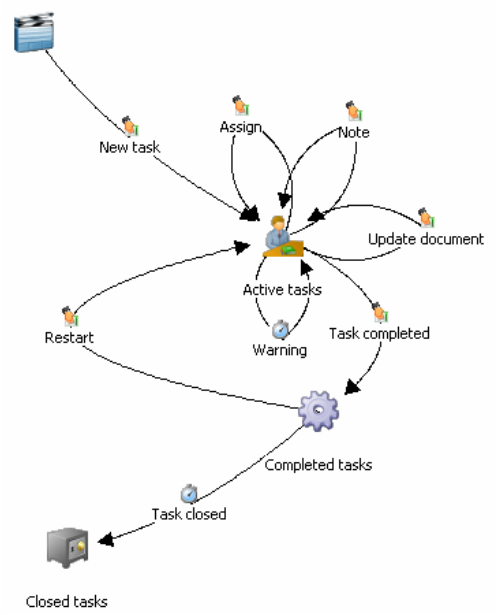


Figure 232: Case Management Sample Procedure

Each person in the team assigned to the task, undertakes their work. When they are finished, they either refer it to another person in the team or declare the task complete.

Any member of the team can add a note to the task.

Preparation

There are two steps necessary for preparing this procedure for use:

1. Register the users who will participate in the procedure. You may use the Set Up procedure to add sample users to your Metastorm BPM system.

There is no need to set up roles, as the roles in this procedure' For Action' and 'Team' are both populated by the process.

2. Add values to the Attributes table to define which users are members of which teams. The attribute name is 'OU'. The Set Up procedure can also be used here to populate the Attributes table. In a real-world implementation, this would typically be populated from and managed in a directory.

Points to Note

- This can be built into the middle of a more structured process if required.

- The ‘Forward’ action makes use of the ‘Rebuild To Do list’ feature. If this were not used, then even though the user has selected a person for the folder to go to next, it would not move from their To Do list onto the list of the person they selected.
- In this example, there is no option to change the team that the item has been allocated to. This example could be amended so that either any team member or a team leader could refer the item to another team.

Flight

This procedure illustrates an approval and booking process for travel requisitions.

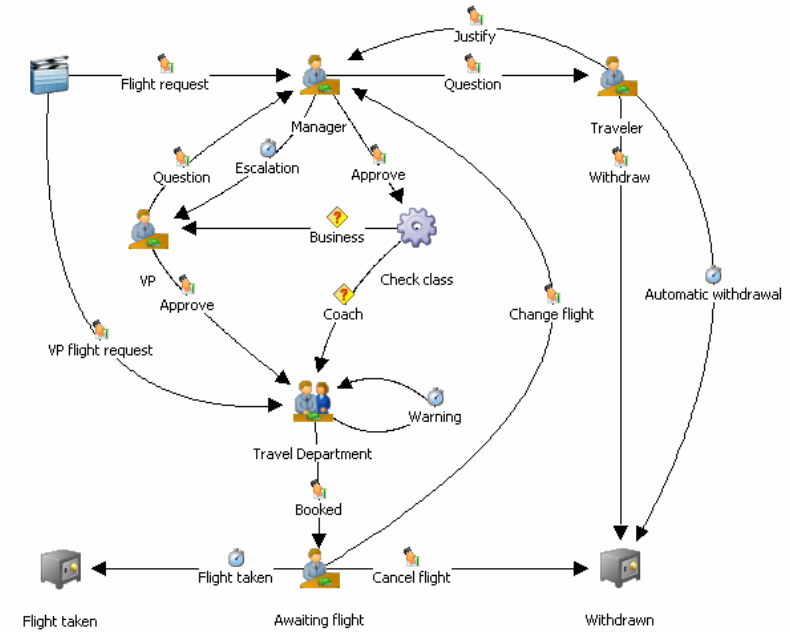


Figure 233: Flight Sample Procedure

Preparation

The following are steps necessary for preparing this procedure for use:

1. Register the users who will participate in the procedure. You may use the Set Up procedure to add sample users to your Metastorm BPM system.
2. If you have used the Set Up Sample Procedure, the necessary information for the table Airport will have been populated. If you have not used Set Up Procedure you will need to populate this table manually.

Points to Note

- This procedure uses dynamic roles that are evaluated at run time based on data collected by the procedure:
  - 'Traveler'
  - 'Manager'
  - 'VP'
  - 'Travel Department'
- Use of the Manager function in the 'Manager' role.
- Use of a database table to populate the 'Airport' drop-down fields on the request form.
- Use of the System stage to route the folders according to the class of travel the traveler wishes to take. Business Class travel has to have a sign off by the VP.

- The definition of the Airport table in the external table editor.
- The use of visibility on the request form to reveal the details field only when ‘Other’ has been selected from the ‘Reason’ drop-down.
- The ‘Progress’ form that displays the process map and indicates the stage the folder is sitting at, whose To Do list it is on, and how long it has been at that stage.
- The separate ‘VP flight request’ creation action, which, for VPs, bypasses the approval stages.

 © Metastorm Inc., 2008

Lead Tracking

This procedure collects sales leads as they are received by an organization and track their progress through to the receipt of an order or the abandonment of the lead.

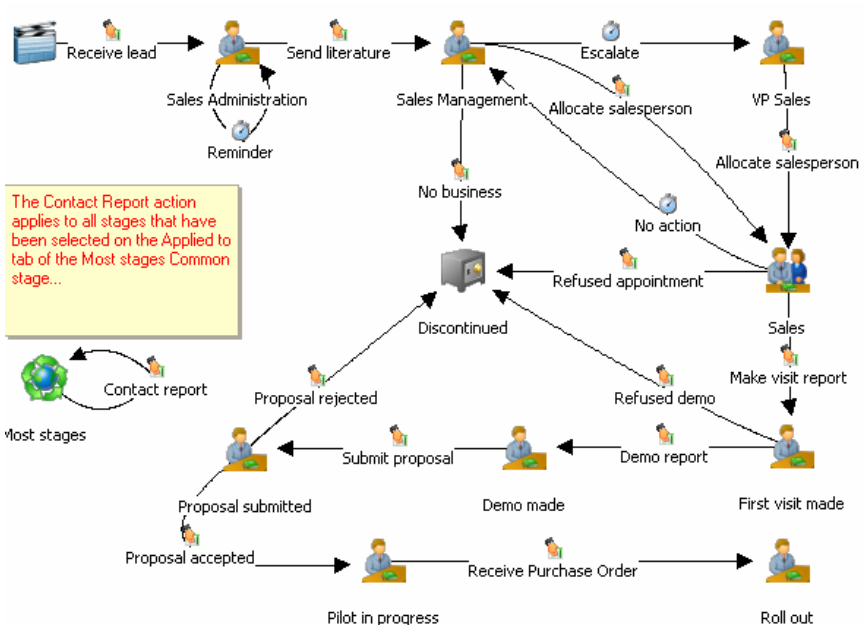


Figure 234: Lead Tracking Sample Procedure

Preparation

The following step is necessary for setting up this procedure:

1. Register the users who will participate in the procedure. You may use the Set Up procedure to add sample users to your Metastorm system.

Points to Note

- This procedure uses the following dynamic roles that are evaluated at run time based on data collected by the procedure:
  - 'Marketing'
  - 'Sales Administration'
  - 'Sales Directors'
  - 'Sales Manager'
  - 'Sales Person'
  - 'Sales VP'
- A key feature of this procedure is the use of a Common stage. The Common stage is called 'Contact Report' and has one loopback action also called 'Contact Report'. (See the Applied to page of the Properties editor for the list of stages where the 'Contact Report' action will be visible. The 'Contact Report' is available to the checked stages.) The use of the Common stage avoids a separate loopback action 'contact report' having to be drawn onto every one of the stages that the 'Contact Report' has been applied to.

The benefit of the use of the Common stage is that the map is less cluttered and easier to understand.

- Other features of note are the use of cascading visibility on the 'New Lead' form:
  - The fields for 'Budget' and 'Sponsor' appear only if the user clicks Yes in the 'Project' field.
  - The 'Budget Amount' field appears only if the user clicks on Yes in the 'Budget' field.
  - The 'Timescale' field appears only if the user clicks Yes in the 'Sponsor' field.

*See the **Visibility** field on the **Extra** page of the **Properties editor** for each of these fields to see how this was established.*

  - The 'Contact Report' form is also useful. This report displays all of the notes and contact reports that have been added to this folder, when each note was added and who added it. This form allows the Sales Manager and others to view the progress of the folder and the notes that have been added to it at each stage.

 © Metastorm Inc., 2008

New Hire

This procedure ensures that the necessary administrative processes are undertaken both before and immediately after a new employee joins an organization. It illustrates the use of Sub-procedures and Rendezvous actions.

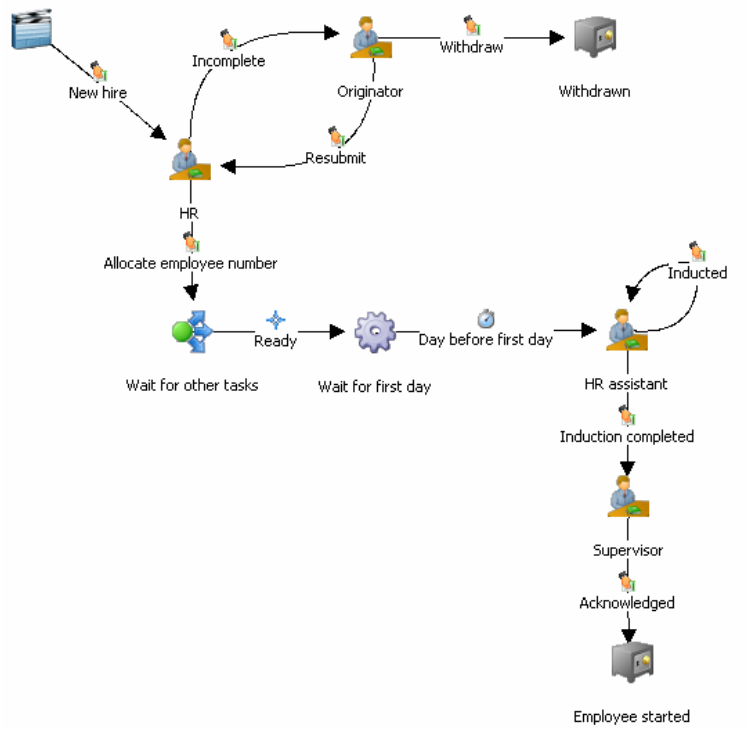


Figure 235: New Hire Sample Procedure (Main Map)

When a folder reaches the 'Wait for other tasks' Sub-procedure stage, child folders are spawned in three sub-maps. When all three have completed their sub-maps the Rendezvous action releases the folder where it sits in the 'Wait for first day' stage.

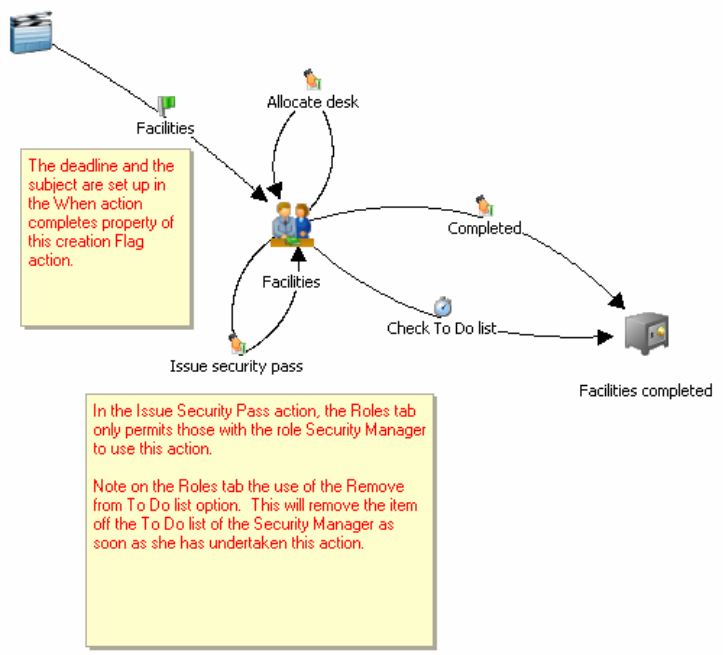

Facilities
Allocate desk
The deadline and the subject are set up in the When action completes property of this creation Flag action.
Completed
Facilities
Check To Do list
Issue security pass
Facilities completed
In the Issue Security Pass action, the Roles tab only permits those with the role Security Manager to use this action.
Note on the Roles tab the use of the Remove from To Do list option. This will remove the item off the To Do list of the Security Manager as soon as she has undertaken this action.

Figure 236: New Hire Sample Procedure (Facilities Submap)

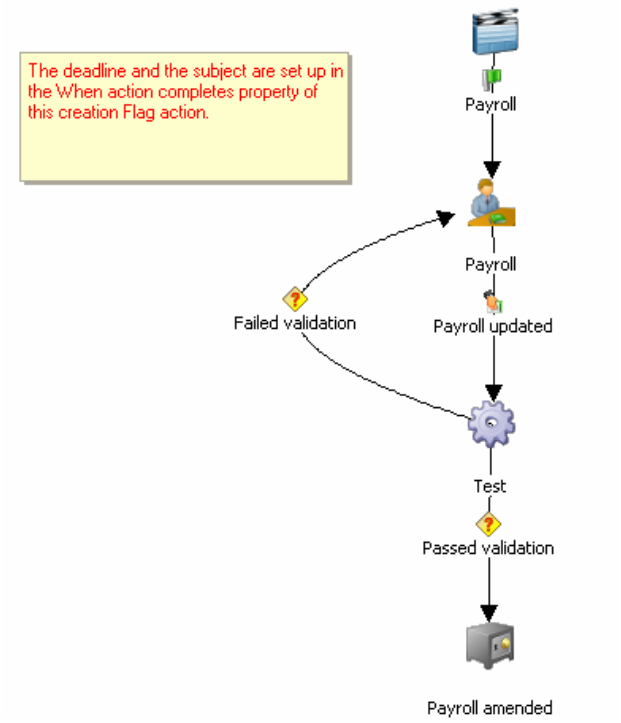


Figure 237: New Hire Sample Procedure (Payroll Submap)

 © Metastorm Inc., 2008

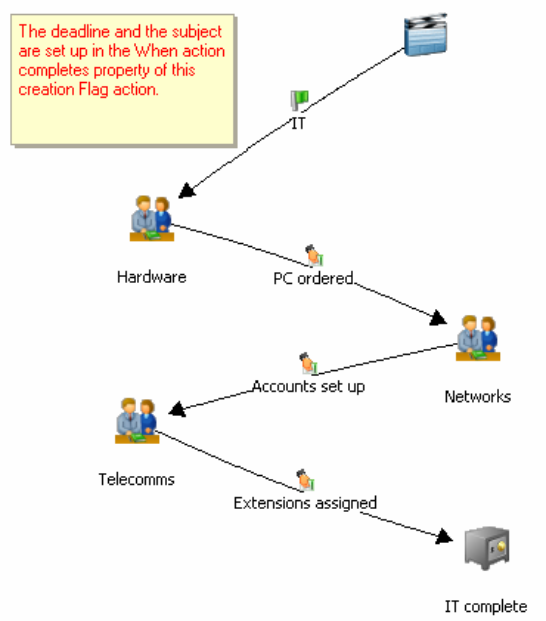


Figure 238: New Hire Sample Procedure (IT Submap)

Preparation

Register the users who will participate in the procedure. You may use the Set Up procedure to add sample users to your Metastorm system.

Points to Note

- This procedure uses the following dynamic roles that are evaluated at run time based on data collected by the procedure:
  - 'Facilities Manager'
  - 'Hardware'
  - 'HR Assistant'
  - 'HR Manager'
  - 'Manager'
  - 'Networks'
  - 'Payroll Clerk'
  - 'Security Manager'
  - 'Telecomms'
- This procedure demonstrates the use of sub-procedures with sub-folders relating to a folder being processed in parallel. When a folder arrives at the 'Wait' sub-procedure stage, three child folders are created in each of three sub-maps: 'Facilities', 'IT', and 'Payroll'. (See the Maps page of the Properties editor for the 'Wait' stage, where all three of the Map check boxes have been checked.)
- The parent folder is released from the sub-procedure stage by the 'Ready' Rendezvous action. All three of the child folders must reach 'Archive' stages in their respective maps for the Rendezvous action to start. (See the Maps page of the Properties editor for the

'Ready' action where all three sub-maps have been checked and the option to wait for all sub-maps to complete has been selected.)

- The folders in the three sub-maps ('Facilities', 'IT', and 'Payroll') can be processed in parallel.
- A separate child folder will be created for each sub-map. This procedure illustrates how data from the parent folder can be displayed in a child folder. The deadline fields in the child folders are set equal to the deadline of the parent folders through an assignment in the flag creation action in the sub-map. (See the When action completed field of the Do this page of the Properties editor of the creation flag action.) When an assignment is used, the data is stored permanently in the child folder.
- An alternative method of displaying data from the parent folder in a form associated with a child folder is illustrated in the 'Name' field of the 'Information Technology' form. The 'Name' field is a calculated field populated from the parent folder each time the form is displayed. (See the calculation field of the Do this page in the Properties editor for the 'Name' field.) When a value for the parent folder is displayed through a calculated field using this method, the value is not stored in the child folder, but is refreshed each time from the parent folder. This has the advantage of not duplicating the data storage.
- After the folder has been released from the sub-procedure stage by the Rendezvous action, it is held at the 'Wait for First Day' stage until it is released by the timed action 'First Day'. This releases the folder to the 'HR assistant' stage one day before the folder deadline (which must be set to the start date in the form that is completed when the folder is created).

 © Metastorm Inc., 2008

One Call

This procedure is for a call center that provides a one-stop service for their customers' calls. A designated member of each team takes incoming calls, and closes calls whenever possible. If they cannot close the call immediately, they take details and place the call on a team queue. Team members pick up calls in the queue and process them. If they are unable to close the call before they go off shift, they release the call back to the team queue.

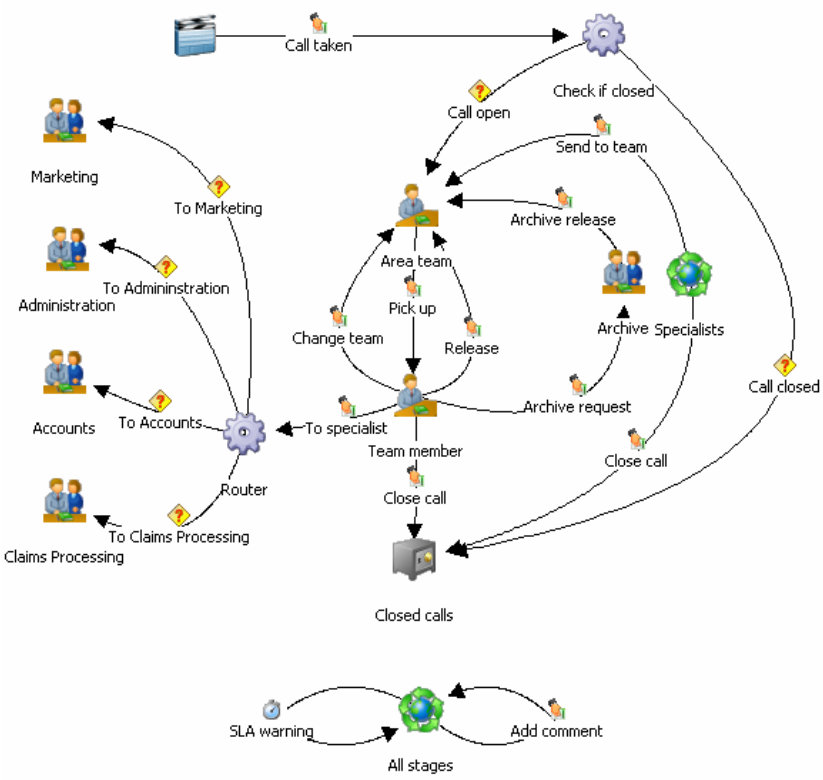


Figure 239: One Call Sample Procedure

Preparation

The following steps are necessary to prepare this procedure for use:

1. Register the users who will participate in the procedure. You may use the Set Up procedure to add sample users to your Metastorm system.
2. Allocate some users to teams. Do this by inserting records in the Metastorm Attributes table. If you have used the Set Up procedure to set up sample users you may use the 'One Call and Case Management' action to populate the attribute table from where the team roles are evaluated. You may populate the table yourself either through the Users and Roles tool or from a directory using the Directory Extraction Service.

Points to Note

- This procedure uses the following dynamic roles that are evaluated at run time based on data collected by the procedure:

  - 'Accounts'
  - 'Administration'
  - 'Archives'
  - 'Claims Processing'
  - 'Marketing'
  - 'AssignedTo'
  - 'Team'
- There are no actions leaving the 'Marketing', 'Accounts', 'Administration' and 'Claims Handling' stages. The actions to move folders out of these stages can be found in the Common stage, 'Specialists'.
- Note how the call history in this procedure is displayed on the 'Call Details' form.

 © Metastorm Inc., 2008

Purchase Order

This procedure demonstrates the use of an editable grid to implement a typical 'master and multiple detail line' form.

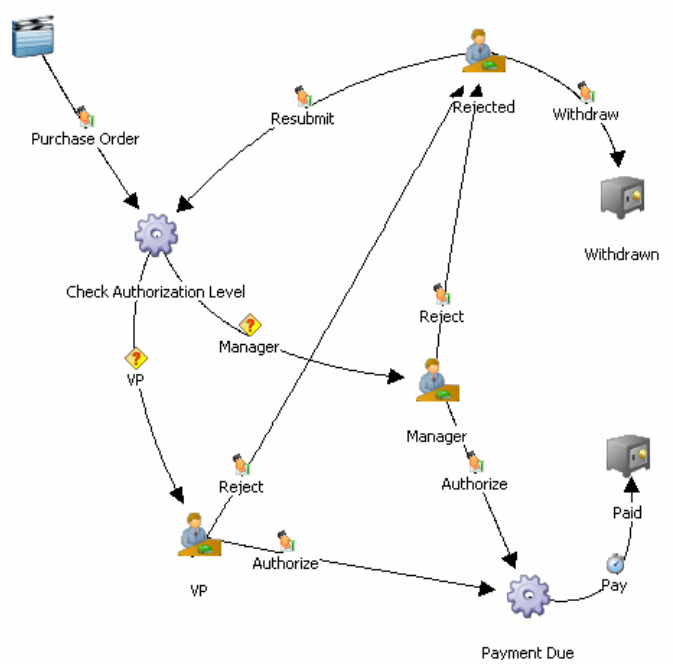


Figure 240: Purchase Order Sample Procedure

Preparation

Once published, this procedure requires a management hierarchy to be set up and populated. This may be done using either the Set Up Sample Procedure, or the Users and Roles tool.

The procedure also requires a set of Purchase Order Item Codes to be set up. This may be done using the Set Up Sample Procedure, or the 'Purchase Order Admin' form (provided as part of the procedure).

Points to Note

As well as the Purchase Order map itself, this procedure provides a 'Purchase Order Admin' administration form, used to maintain the Purchase Order Item Code table.

The procedure makes use of two external tables (set up automatically when you publish the procedure):

- Purchase Order Item stores individual item lines for each Purchase Order
- Purchase Order Item Code stores the available item codes, and associated descriptions and unit prices.

The grid used to enter purchase order item lines makes extensive use of client scripting, for validation and the calculation of line and purchase order totals.

It is not possible to take the Purchase Order procedure offline.

 © Metastorm Inc., 2008

Training

This procedure was written to help the facilitation of training classes.

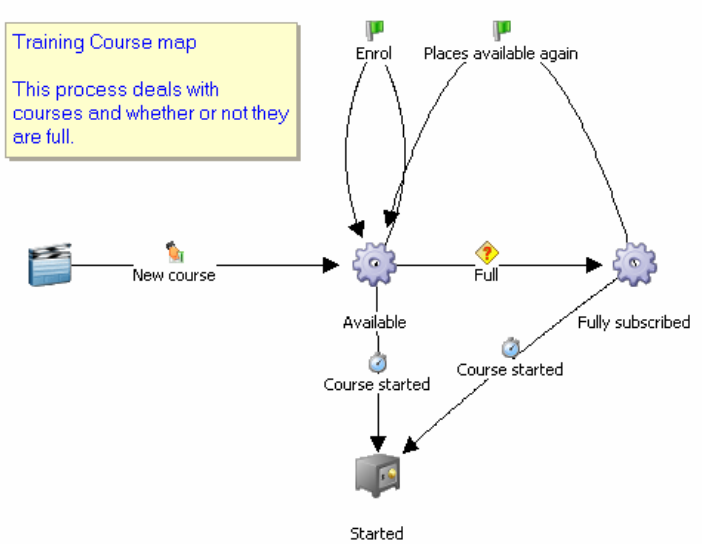


Figure 241: Training Sample Procedure (Training Course Map)

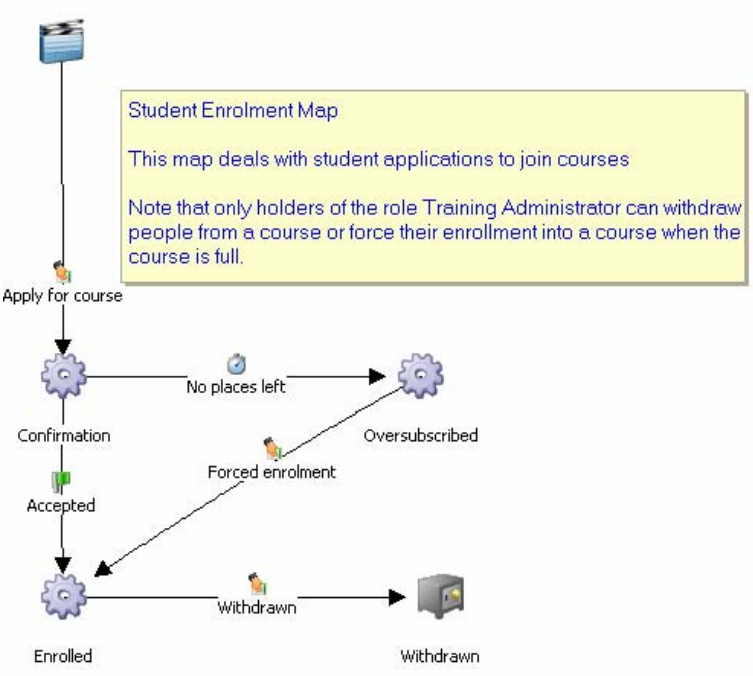

Student Enrolment Map
This map deals with student applications to join courses
Note that only holders of the role Training Administrator can withdraw people from a course or force their enrollment into a course when the course is full.
Apply for course
No places left
Confirmation
Oversubscribed
Forced enrolment
Accepted
Withdrawn
Enrolled
Withdrawn

Figure 242: Training Sample Procedure (Student Enrolment Map)

Preparation

To prepare this procedure for use, register the users who will participate in the procedure. You may use the Set Up procedure to add sample users to your Metastorm system.

Points to Note

Although at first glance this appears to be a simple procedure using maps with only a very few stages and simple forms, it is quite advanced in a number of respects. The procedure uses two maps: Training Course and Student Enrolment. A single procedure would use multiple maps when distinct types of data are being collected.

- Training Course

  The Training Course map allows the Training Administrator to set up a new class starting on a particular date, for example, an Economics class starting on 1 September. This map would create a new folder for each subject/date combination. Each folder would contain four specific pieces of information:
  - the subject
  - the date the class starts
  - the maximum number of students that may be enrolled in the class
  - the actual number of students that have been enrolled in the class

  The form could be extended to include details about the location, the time and the teacher.

- Student Enrolment

  The Student Enrolment map handles the enrolment of students into the classes. This map creates a new folder for each student that attempts to register for a class. If a student attempted to register for two classes, there would be two folders for that student. These folders would store three items of information:
  - the name of the student
  - the class they wish to enroll in
  - the date the desired class begins

  Folders are created and stored for all students attempting to enroll in a class whether they are successful or not.

- The second advanced feature in this procedure is the use of flags to signal actions between the two maps. When a student attempts to enroll in a class, the 'Apply for course' action raises a flag to notify the Training Course map that it needs to know whether there is space in the class to accept the student. That folder then waits in the 'Confirmation' stage for the request action to raise a flag to signify that there is space in the class.
- This signaling between folders also makes use of the Metastorm parent/child folder relationship. Each student enrolment folder assigns the relevant class folder as its parent. (See the **When action completed** command of the **Apply for course** action). This is subsequently used to ensure that the correct class is checked when inquiring whether there are sufficient spaces to enroll the student.
- Note that this procedure uses the following dynamic roles which are evaluated at run time based on data collected by the procedure:
  - 'Training Administrator'

 © Metastorm Inc., 2008

- 'Training Department'

Wizard

This procedure demonstrates the use of chained actions to provide a wizard-style interface.

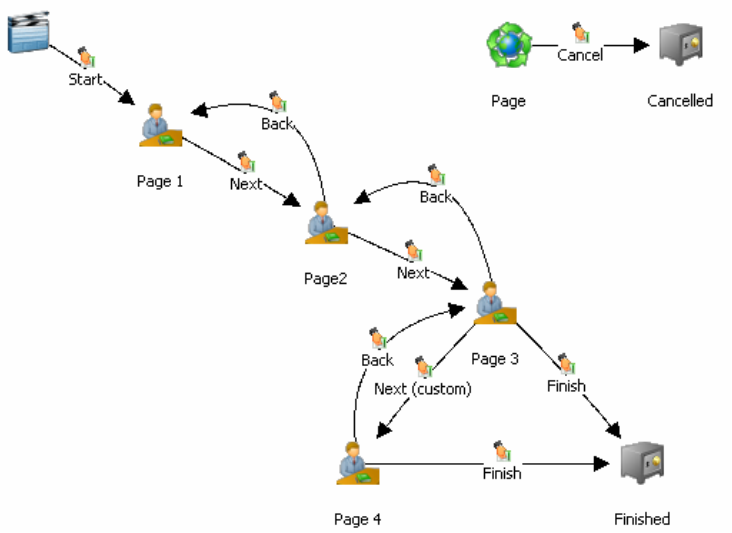


Figure 243: Wizard Sample Procedure

Preparation

No special preparation is required for this procedure.

Points to Note

As each action is completed, the next action to be performed is evaluated and opened automatically.

If the user cancels at any stage, the folder is left on their To Do list for subsequent completion.

 © Metastorm Inc., 2008

BRI

This procedure illustrates the use of Metastorm BPM with a business rule engine.

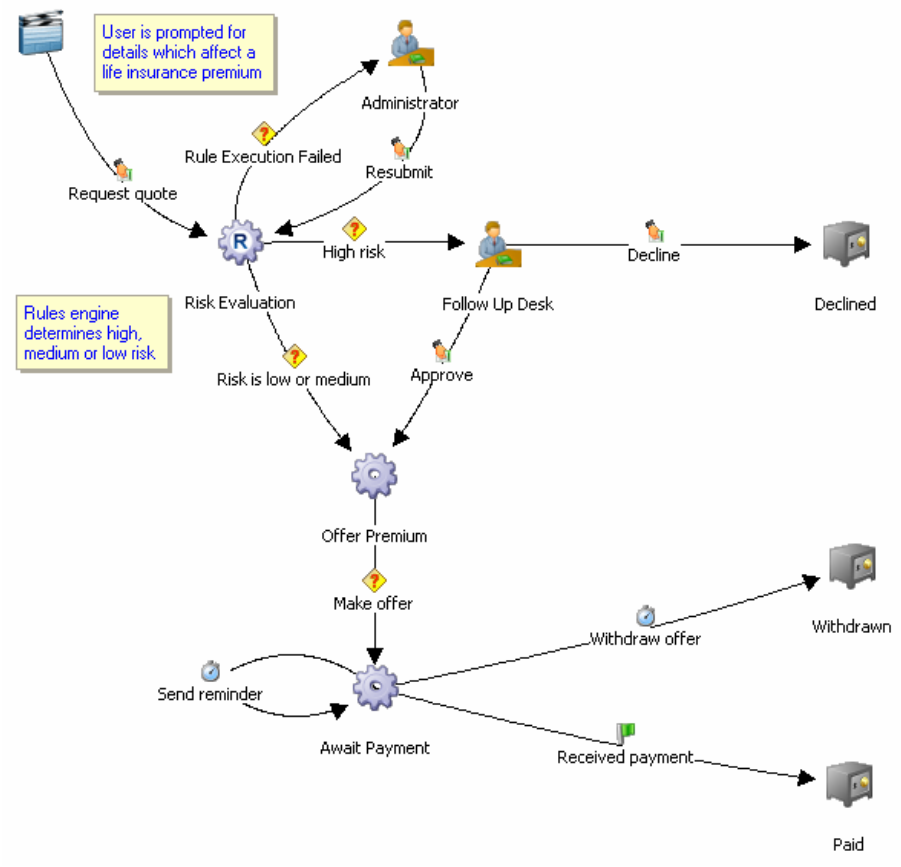


Figure 244: BRI Procedure (Policy Quote Map)

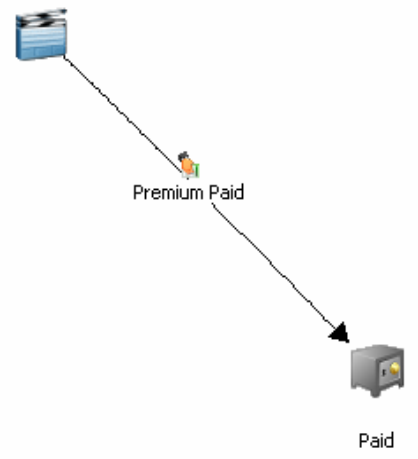


Figure 245: BRI Procedure (Premium Payment Map)

Preparation

In addition to the Sample Library, this procedure requires you to publish the BRI library.

In order for the procedure to work, access is required to:

- a business rules engine.
- a Rule Engine Adapter web service such as the BizTalk Rule Engine Adapter provided with Metastorm BPM.

A set of rules must have been defined, using the rule designer for the business rules engine you are using.

Points to Note

A key feature of this procedure is the use of a Rules stage, which allows users to execute a rule.

To use the procedure:

1. Use the BizTalk Rule Deployment Wizard to deploy the BRI Insurance Policy.xml file.
2. After publishing the procedure, ensure that a user has the Insurer role.
3. On the rules stage, you may need to edit the **Rule Connection** string to specify a different machine on which the Metastorm Rules Service is installed.
4. If the rule execution fails or times out, the administrator stage allows an administrator to resubmit a quote. The event log or designer log may explain why the rule failed.

 © Metastorm Inc., 2008

External PO

This procedure demonstrates the integration with BizTalk Servers offered by the Metastorm Accelerator for BizTalk.

It is based on a tutorial provided by Microsoft at:
http://msdn.microsoft.com/library/en-us/dnbiz/html/learnbtsanchor.asp

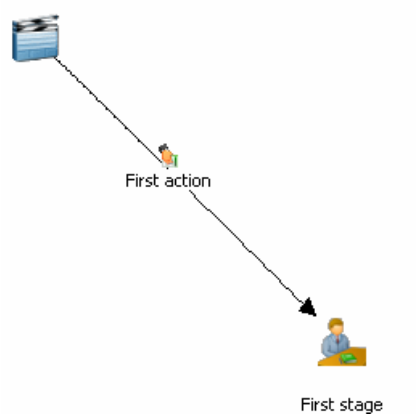


Figure 246: External PO Procedure

Points to Note

After installing the BizTalk sample:

1. Set up a partnership agreement using the Metastorm B2Bi admin procedure.
2. From the BizTalk Document Editor, open the *ReceivedPOSchema.xml* file and publish it the BizTalk server's WebDav repository.
3. From the BizTalk mapping tool, open the *E-POMap.xml* file and save it to the BizTalk server's WebDav repository.
4. From the BizTalk Messaging Manager, modify the Channel To Contoso so that the outbound document points to *Metastorm PO.xml* and the mapping points to *E-POMap.xml*.
5. From the same tool, modify the Contoso Port to point to an Accelerator URL (e.g. http://localhost/scripts/eB2BListener.dll/ID=<PartnerID>&&Map=Received+PO).

To use the procedure, copy the *ReqToApprove.xml* file from the documents to pickup folder.

For further details, refer to Lesson 2 of the LearnBiztalk.exe documentation.

For documents from Metastorm BPM to BizTalk server, use on of the three ASP files provided (*eworkToXLANG.asp*, *eworkToFile.asp* and *eworkToMSMQ.asp*).

Send Mail

This procedure demonstrates the sending of mail to an SMTP server or a folder. It requires the Mail library.

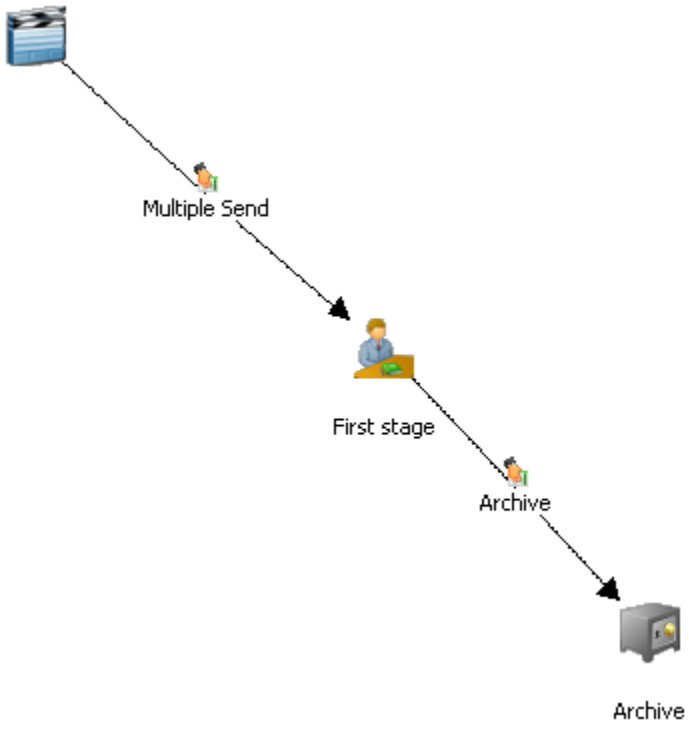


Figure 247: Send Mail Procedure

Points to Note

The following fields are available on the Mail form:

- **To** addresses, separated by semicolons.
- **CC** addresses, separated by semicolons.
- **BCC** addresses, separated by semicolons.
- **Subject**.
- **Text** or **HTML**.
- **Sender**.
- **Attachments**, separated by semicolons.
- **Pickup** folder to which to send the mail.

 © Metastorm Inc., 2008

News

This procedure provides, for all users, a process for maintaining a discussion database, comprising a list of topics and associated note threads, along with an Administration Form for browsing available topics by category.

Users can create a new topic, add a note to an existing topic, or simply view all notes for any existing topic. The originator, or an Administrator, can close any topic at any time.

Appendix B - Formula Components

Formulas should not be confused with client extensions. The Process Engine evaluates formulas, whereas the Client evaluates client extensions. This appendix does not address client extensions.

Type-face Conventions

Throughout this appendix, the following type-face conventions are used:

- *formula data types* are shown in italics
- ***function arguments*** and ***element properties*** are shown in bold italics
- **[optional arguments]** are shown in brackets
- **{** optionally repeated arguments **}** are shown in braces
- **`code examples`** are shown in bold monotype (with comments in `// regular monotype`).

General Syntax

Metastorm formulas are composed of the following types of elements:

- Variables – these may be built-in variables or custom (folder) variables, which are defined within Designer for each map
- Field data
- Function calls
- Operators
- Literal values
- Comments

Comments comprise any text preceded by a double slash (`//`), up to and including either the end of the formula as a whole, or a new line. Comments are stripped out by the Metastorm Designer's 'Publish' function, and are not stored into the Metastorm database or interpreted by the Process Engine.

Metastorm BPM recognizes any string of letters, digits and underscores, starting with a letter and preceded by a single percent character (`%`), as either:

- A function call, if the string is immediately followed by an opening parenthesis (`(`). Any following strings up to a matching closing parenthesis (`)`) are interpreted as function arguments. Individual arguments are separated by commas (`,`). Arguments containing empty strings may be specified explicitly using double quotes (`"`), or simply omitted. However, the comma separators used between arguments should never be omitted, except where those arguments are explicitly shown as optional.
- An element of a built-in array variable, if the string is immediately followed by an opening square bracket (`[`). Any following string up to a matching square bracket (`]`) is interpreted as an index into an array variable. An array variable is any variable that holds a number of tab-delimited substrings, with the first substring being that with an index value of zero.
- A system variable, if the string contains a period (`.`).

 © Metastorm Inc., 2008

- A folder variable, otherwise.

Variable and function names are not case sensitive: **%FolderName**, **%FOLDERNAME** and **%foldername** all refer to the same variable.

Any other string is interpreted as either an operator or a literal value. Operators may be any of the following:

- Unary (prefix) operators, which alter the meaning of any following formula component:

-	make following number negative
~	negate following condition
%	treat the following character as a literal, rather than an operator
(boolean)	convert following formula component to a different type
(integer)	
(real)	
(date)	

- Binary (infix) operators, which perform some operation on the formula components to either side:

arithmetic	**+**, **-**, ***** and **/**
Boolean logic	**=**, **>**, **<**, **>=**, **<=**, **<>**, **&** and **|**
assignment command	**:=**
command separator	**;**

- A tertiary (infix) operator, which:

Tests the first (condition) formula component before **?**
If true, evaluates the second formula component (after **?** but before **:**)
Otherwise evaluates the third and final formula component (after **:**)

- Grouping (outfix) operators:

()	parentheses, used to control the order of evaluation of arithmetic, logical and conditional operators, with nested terms being evaluated before outer terms
""	matched pairs of quotation marks

Operators are covered in greater depth in the final section of this document.

Formula Types

Any formula, and any term or argument within a formula, has a type. This may be any of:

- number: a numeric value. This may be a literal (e.g. **`77`**), a (real or integer or currency) variable (**`%Priority`**), a function result (**`%Length(a string)`**), or the value of an arithmetic expression itself composed of numeric values (**`%Priority + 1`**) or a conditional expression which returns numeric values (**`%Priority > 0 ? %Priority : 10`**).
- string: a text string. This may be a literal (quoted or not), a (text or memo) variable, a function result, a conditional expression, or any combination of these (e.g. **`This folder is called %FolderName`**). A maximum length (which may be up to 250) may be specified for any string property or variable; no maximum length is specified for a function argument or result. Any number may also be used as a string: it will be implicitly converted by Metastorm BPM.
- list: a text string comprising any number of comma-separated string values. These may be entered as successive lines, rather than comma-separated, in properties. Where a list is passed as an argument, it must be quoted to avoid the function treating it as a sequence of arguments.
- date-time: a date and time value. This may be a (time) variable, a function result, a conditional expression, or any combination of these. The Designer does not provide a way to enter a literal date-time; nor does Metastorm BPM perform any implicit conversion of other values to date-time values.
- condition: a logical condition. This may be a literal (**`TRUE`** or **`FALSE`**) value, a (check) variable, a function result, a logical or conditional expression, or any combination of these. Any number may be used as a condition: **`0`** will be treated as **`FALSE`**, and any other value as **`TRUE`**.

Some properties expect one or more *commands*. This is notionally a valueless property, used for its side effects. This may be a function call, an assignment, a conditional expression, or any sequence of these. C*ommands* are actually implemented as *conditions*, which return **`TRUE`** on success and **`FALSE`** otherwise. This means that the operation of a command may be checked in any condition, and different paths taken depending on its success or otherwise.

Type Conversions

Metastorm BPM will perform implicit type conversions, when property, term or argument returns a result of one type, but the property, term or argument expected is of another type. Implicit type conversions are performed as follows:

- string to list: a single element list, unless the string contains commas, in which case each comma will be treated as a list item separator.
- string to number: leading spaces are ignored. Any leading numeric (integer or real, with the locale-independent period (`.`) as the decimal point) part of the string is then converted to a number, and any remaining part of the string is ignored. If there is no leading numeric part to the string, the result will be a zero.

 © Metastorm Inc., 2008

- string to date-time: leading spaces are ignored. The remainder of the string is then converted to a date-time, using the server's locale settings. If the string does not contain a valid date-time (according to the server's locale), the conversion will fail.
- string to condition: True and False (in any combination of case) are converted to **TRUE** and **FALSE**, respectively. With any other value, the conversion will fail.
- list to string: the entire contents of the list, including all comma separators, are treated as a single string.
- list to date-time: proceeds via list to string and then string to date-time.
- list to number: proceeds via list to string and then string to number.
- list to condition: proceeds via list to string and then string to condition.
- number to string: the minimal decimal representation of the number (with decimal places if required). This conversion is locale-independent: it will always use a period (**.**) as the decimal separator (if required), and will never use any thousands separator. To specify a different string representation of a number, use the **%FormatNumber** function.
- number to list: proceeds via number to string and then string to list.
- number to condition: zero for **FALSE**, any other value **TRUE**.
- date-time to string: uses the short date and time format from the server's locale settings. To specify a different string representation of a date-time, use the **%FormatTime** function.
- date-time to number: proceeds via date-time to string and then string to number.
- condition to string: True or False.
- condition to list: proceeds via condition to string and then string to list.
- condition to number: zero for **FALSE**, **-1** for **TRUE**.

Attempts to perform the following conversions will always fail:

- number to date-time.
- date-time to condition.
- condition to date-time.

A number of explicit type conversion operators are provided, which are useful for forcing type conversions when Metastorm BPM might otherwise suppose a type conversion is not required. These follow the same rules as for implicit conversions:

- **(boolean)** attempts to convert its succeeding formula to a condition.
- **(integer)** attempts to convert its succeeding formula to a number, rounding any digits after a decimal point.
- **(real)** attempts to convert its succeeding formula to a number, maintaining any digits after a decimal point.
- **(date)** attempts to convert its succeeding formula to a date-time.

Strings and Quotes

Strings do not generally need to be quoted, as any string that is not recognized as any of the above will be treated as a text literal. However, there are certain circumstances where quotes are needed, such as when a string containing commas needs to be passed as a single argument to a function call. For instance, a typical SQL **SELECT** statement will itself include commas and

('quoted') literal text values. In order to pass such a statement as an argument to a Metastorm function, the whole string should be "quoted" to prevent Metastorm BPM from trying to interpret the commas in the SQL statement itself. Any quotation marks needed within the **SELECT** statement itself should be written as single quotes ('), which Metastorm BPM treats as literal characters. Note that Metastorm BPM will still evaluate Metastorm variables, functions or other operators within any "quoted" (or any 'quoted') text.

When you are assigning a date time value to a variable, it must be enclosed in quotation marks. E.g. %DateTime1:="2001-11-23T00:00:00".

In general, Metastorm BPM will remove leading and trailing spaces from any unquoted or "quoted" string.

Where Formulas May Be Used

Formulas may be used to set any Metastorm Designer property that has an associated Integration wizard or Formula editor button. These properties, with their expected types, are:

- Map **Subject** accepts a string. Values longer than 250 characters will be truncated at run-time.
- Stage **When stage started** accepts one or more commands, entered on successive lines.
- Stage (but not archive stages) **When stage completed** accepts one or more commands, entered on successive lines.
- Stage (system stages only) **Auto-forward folders to** accepts a string that should, at run-time, resolve to the name of another stage in the same map.
- Rules accepts a string that represents a business rule.
- Action **Alert message** accepts a string. Values longer than 250 characters will be truncated at run-time.
- Action (timed actions only) **Start this action** accepts a number.
- Action (timed actions only) **Timed event** accepts a date-time.
- Action (user actions only) **When action started** accepts one or more commands, entered on successive lines.
- Action (but not user actions) **Only invoke action if** accepts a condition.
- Action **When action completed** accepts one or more commands, entered on successive lines.
- Form **When user loads form** and **When user saves form** both accept one or more commands, entered on successive lines.
- Button (server operations only) **When button pressed** accepts one or more commands, entered on successive lines.
- Button (new folder only) **New folder** accepts a string that should, at run-time, resolve to the name of a published map.
- Button (open folder only) **Open folder** accepts a string that should, at run-time, resolve to the ID of an Metastorm folder.
- Clip (calculated only) **Calculation formula** accepts a string that should, at run-time, resolve to the name of an existing folder attachment.
- Check (calculated only) **Calculation formula** accepts a condition.

 © Metastorm Inc., 2008

- Currency (calculated only) **Calculation formula** accepts a currency.
- Number (calculated only) **Calculation formula** accepts a number.
- Date/Time (calculated only) **Calculation formula** accepts a date-time.
- Text, Drop-down, Radio group, Memo and Signature (calculated only) **Calculation formula** all accept a string.
- List (calculated only) **Calculation formula** accepts a list.
- Clip, Check, Currency, Number, Date/Time, Text, Drop-down, Radio group, List, Memo and Signature (but not calculated) **When changed** accepts one or more commands, entered on successive lines.
- List, Drop-down **List options** accepts a list.
- Date/Time **Earliest** and **Latest** both accept date-times.
- Grid **Row(s)** and Dataset **Row** accept (via SQL dialog) a condition.
- Grid **When user selects row** accepts one or more commands, entered on successive lines.
- Status **Value** accepts a number
- Status **Safety threshold** accepts a number
- Status **Danger threshold** accepts a number
- Status **Action** accepts a string that represents the name of an action form
- Status **Map** accepts a string that represents the name of a map
- Roles (via Add and Edit dialogs) **Formula** accepts a list that should, at run-time, resolve to a list of Metastorm roles and/or user IDs.

Formulas created for dynamic roles are treated as literal strings and may require the use of the %Evaluate function. Refer to the section on Defining Dynamic Role Formulas for further details.

- Flags (via Add and Edit dialog) **Data** accepts one or more strings, entered on successive lines.

Metastorm formulas may also be used within any map script. However, these scripts may themselves be called only by a formula specified for one of the above properties.

When Formulas Are Evaluated

This subsection details the order in which formulas are evaluated.

When a Blank Forms list or Admin Forms list is requested, the **Available to roles** property is evaluated for each action in the list.

When a folder is requested:

1. For each possible action from that stage, the **Available to roles** property is evaluated.
2. For each form marked as available at the stage, the **Available to roles** property is evaluated.
3. For the form that will be displayed, the **When user loads form** property is evaluated.
4. For all the fields on the form, in tab order:
    a. The field's data value (calculation formula or custom variable) is evaluated.

b. If the field is:

- a grid, the grid layout is evaluated.
- not a grid, if the field has an options list (for example, **Earliest**, **Latest** or a Dropdown's list), the options list is evaluated.

When an action is performed:

1. If the action is an action that starts a map, the map's **Subject** is evaluated.
2. The action's **Available to roles** property is evaluated.
3. The action's **When action started** property is evaluated.
4. If the action has a form:
   a. The form's **Available to roles** property is evaluated.
   b. The form's **When user loads form** property is evaluated.
5. If the action request indicates that it should be committed immediately
   a. All fields on the form, in tab order, are updated, based on request data.
      - For all the fields on the form, in tab order:
   b. The field's data value (calculation formula or custom variable) is evaluated, to generate field output.
6. If the field is:
   - a grid, the column list is evaluated.
   - not a grid, if the field has an options list (for example, **Earliest**, **Latest** or a Dropdown's list), the options list is evaluated.
7. If the action request indicates that it should be committed immediately:
   a. The action's **When action completed** property is evaluated.
   b. The **Action to Stage** is evaluated.
   c. The action's **Chained action** property is evaluated.
   d. The action's **Alert message** property is updated.
   e. If the action is leaving a stage, the stage's **When stage completed** property is evaluated. The new stage's **When stage started** property is evaluated.
   f. For each timed action, the action's **Timed Event** property is evaluated.
   g. Any flag actions leaving that stage have their flag name evaluated.
   h. If the action is not a loopback action, or not a 'remove from To Do list' action, the To Do list and Watch list are evaluated.
   i. The action's **Raise flag** property is evaluated.

When a refill is requested:

8. If the refill is for a folder, the folder's **Available to roles** property is evaluated.
9. All fields, in tab order, that have a custom variable are updated.
10. For the fields that caused the refill request, the **When Changed** property is evaluated.

 © Metastorm Inc., 2008

11. For all dependent fields, in tab order:
    a. The field's data value (calculation formula or custom variable) is evaluated.
    b. If the field is:
        - a grid, the column list is evaluated.
        - not a grid, if the field has an options list (for example, **Earliest**, **Latest** or a Dropdown's list), the options list is evaluated.

When an action is submitted:

1. If the folder has a form:
    a. All fields that have a custom variable are updated, in tab order.
    b. The form's **When user saves form** property is evaluated.
2. If it is not an Admin form:
    c. The action's **When action completed** property is evaluated.
    d. The **Action to Stage** is evaluated.
    e. The action's **Chained action** property is evaluated.
    f. The action's **Alert message** property is updated.
    g. If the action is leaving a stage, the stage's **When stage completed** property is evaluated. The new stage's **When stage started** property is evaluated.
    h. For each timed action, the action's **Timed Event** property is evaluated.
    i. Any flag actions leaving that stage have their flag name evaluated.
    j. If the action is not a loopback action, or not a 'remove from To Do list' action, the To Do list and Watch list are evaluated.
    k. The action's **Raise flag** property is evaluated.

*If you are using Process Events Integration, the **Delegate all external events for map** checkbox defines whether all the events of a process will be delegated to an external .NET assembly. The events will be evaluated before or after the actions in the **Do this** tab depending on whether or not the **Perform external events before local events** is checked.*

*To use the **Delegate all external events for map** check box the **Process Events Library** needs to be associated with the current procedure.*

Evaluation of 'When…' Formulas

Metastorm Designer makes available several '**When…**' event formulas, for example, **When User Loads Form**, **When Action Started**.

Forcing Failure

To force any of these formulas, when evaluated, to fail, ensure that the final line of the formula evaluates to False. No error will be reported in the Designer log.

Note that this is different to %User.Error which forces a formula to fail at any point in the evaluation.

The following formulas would fail:

Formula 1:	False
Formula 2:	True False
Formula 3:	%A := 32 %A = 31

Common Stages

When a common stage is applied to any stage, the common stage's formulas are applied at the end of that stage's existing formulas.

For example, if Formula 3 above is in the When Stage Started event of a user stage, and a common stage with the When Stage Started formula:

```
%A=32
```

is applied to it, the evaluation no longer fails, because the formula evaluated for the user stage is:

```
%A :=32
%A = 31
%A = 32
```

So, the final line evaluates to True.

System Variables

Metastorm system variables are used to hold data specific to:

- the Metastorm Process Engine
- the Metastorm user on whose behalf the Process Engine is performing a transaction
- a Metastorm (user or system) action which the Process Engine is performing.

All Metastorm system variables are predefined; new system variables cannot be defined from within Designer.

%Action.Name

string	**read-only** name of the currently executing action, as defined in the procedure map.

 © Metastorm Inc., 2008

This variable is populated when an action is started on a folder and cleared when the action is completed. The field may not be cleared if the action form is closed without the action being submitted or cancelled. For example, the browser is closed without the action being completed.

This variable should **not** be used in the following properties:

- Stage **Auto-forward folders to**, since this is evaluated by a subsequent hidden action (which is always named =>).
- Field **Variable**, since the variable is read-only.
- Roles **Formula**, since it may have a different value every time the formula is evaluated.

%Action.Name is not displayed on a folder page. However, if a folder is locked by another user who is performing an action, the %Action.Name is displayed on a folder form.

%Action.Notes

string **read-write** memo to be written to the selected folder's audit trail, if and when the current action is committed. Until and unless assigned to, it will be empty. The result may subsequently be read back from the audit trail (in a folder or action form) using the (**read-only**) **`%Notes`** array variable.

When used in a form, this variable will always be empty when the form is shown in a folder. It should **not** be used in the following properties:

- Stage **Auto-forward folders to**, since this is evaluated by a subsequent hidden action.
- Roles **Formula**, since it may have a different value every time the formula is evaluated.

%Action.StartsStage

string **read-only** name of the stage that the current action will move the selected folder to. This is normally the target stage of the action, as drawn in the procedure map. If the action is a loop-back, then this variable will hold an empty string. If the target is a system stage, with its ***Auto-forward folder to*** property set, then this variable will hold the name of the stage to which the folder will be auto-forwarded.

This variable should be used **only** in the following properties:

- Stage When stage started.
- Stage When stage completed.
- Action Alert message.
- Timed action Start this action and Timed event.

%Form.Name

The function %Form.Name returns the form name when viewed in an action, refill or submit request. This function returns a value in a folder. For example, the form name is returned where a user has started an action and the action form is displayed.

string **read-only** name of the current form.

This variable should be used **only** in the following properties:

- Form When user loads form.
- Field Calculation formula.
- Field List options.
- Field When button pressed, When field changed and When user selects row.
- Form When user saves form.
- Roles when a role is used in the 'Restrict Viewing to roles' property of a form.

The function %User.Form contains similar functionality except it returns an empty value in a folder.

%Procedure.Name

string **read-only** name of the current procedure.

%Procedure.Version

string **read-only** version number of the current procedure.

%Session.FlagName

string **read-only** name of the flag that triggered a flagged action. This will be empty for any other kind **only** of action.

This variable should be used in the following properties:

- Map **Subject** (but only if the map is initiated by a flagged action).
- Flagged action **Only start action if**. Note that using this property is considerably less efficient at run-time than specifying a **Folder which must raise the flag**, if the intention is to check the value of **`%Session.FlagRaiser`**.
- Flagged action **When completed**.
- Flagged action **Alert message**.

%Session.FlagData[column_number]

string **read-only** contents of the specified column of flag data, in a flagged action. The first column (which contains **`%Session.FlagName`**) is column 0. This will be empty for any other kind of action.

This variable should be used **only** in the following properties:

- Map **Subject** (but only if the map is initiated by a flagged action).
- Flagged action **Only start action if**. Note that using this property is considerably less efficient at run-time than specifying a **Folder which must raise the flag**, if the intention is to check the value of **`%Session.FlagRaiser`**.
- Flagged action **When completed**.
- Flagged action **Alert message**.

 © Metastorm Inc., 2008

%Session.FlagRaiser

string **read-only** ID of the folder which raised the flag that triggered a flagged action (or of the folder specified in the Raise Flag call). This will be empty for any other kind of action.

This variable should be used **only** in the following properties:

- Map **Subject** (but only if the map is initiated by a flagged action).
- Flagged action **Only start action if**. Note that using this property is considerably less efficient at run-time than specifying a **Folder which must raise the flag**, if the intention is to check the value of **`%Session.FlagRaiser`**.
- Flagged action **When completed**.
- Flagged action **Alert message**.

%System.Name

string **read only**. When the Engine starts, a database lookup is performed against the eServer table. %System.Name is the cached value of the eServerName column.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%System.Roles

list **read-only** list of all roles defined in the Metastorm database

This variable is evaluated via a database lookup, and thus may have an effect on performance. It should **not** be used in the following properties:

- Field **Variable**, since the variable is read-only.
- Stage Auto-forward folders to.

%System.Time

date-time **read-only** current date and time, as the Process Engine evaluates the formula.

This variable should **not** be used in the following properties:

- Field **Variable**, since the variable is read-only.
- Stage Auto-forward folders to.

%System.Users

list **read-only** list of all users registered in the database

This variable is evaluated via a database lookup, and thus may have an effect on performance. It should **not** be used in the following properties:

- Field **Variable**, since the variable is read-only.

- Stage Auto-forward folders to.

%User.Error

string **read-write** custom error message. Until and unless assigned to, it will be empty.

Setting this variable causes the commit to fail and the error message to be given to the user.

This variable should be used **only** in the following properties:

- Field **Variable** (allowed, since the variable is read-write, but of dubious value).
- Field **Calculation formula** – this is typically how validation error messages are displayed on screen.
- Field **When button pressed**, **When field changed** and **When user selects row** – these are typically where validation error messages are generated.

%User.Form

The expression %User.Form returns the form name when viewed in an action, refill or submit request. For example, where a user has started an action and the action form is displayed.

string **read-only** name of the current form.

This variable should be used **only** in the following properties:

- Form When user loads form.
- Field Calculation formula.
- Field List options.
- Field When button pressed, When field changed and When user selects row.
- Form When user saves form.
- Roles when a role is used in the 'Restrict Viewing to roles' property of a form.

This function returns an empty value in a folder. The function %Form.Name also returns the form name, however (unlike %User.Form) it does so in all situations, including in a folder.

%User.Input

string **read-only** value just entered by the user into a data field, and not yet validated.

This variable should be used **only** in the **When field changed** property.

%User.Input[column_number]

string **read-only** contents of the specified column of the row just selected by the user from a grid. The first column in the row is column 0.

This variable should be used **only** in the following property:

- Grid field **When user selects row**. At this time, **`%User.Input`** will contain all the columns of the selected row, as a tab-separated array.

 © Metastorm Inc., 2008

%User.Name

string **read-only** registered user ID of the current user.

This variable should **not** be used in the following properties:

- Field **Variable**, since the variable is read-only.
- Stage Auto-forward folders to.

Folder Variables

Metastorm folder variables are used to hold data specific to the folder on which the Process Engine is performing a transaction. Folder variables may be either predefined, or 'custom' variables defined for each map using the Designer.

Except in the course of an action, all folder variables are **read-only**. Where specified, they are **writeable** within the context of an action.

Although folder variables may be used in any Roles ***Formula***, they should not be used in any role used to control access to a (user) creation action, since this is evaluated in order to retrieve a Blank Forms list, without access to any individual folder.

%ActionCount

number **read-only** number of actions committed thus far on the selected folder. This will be zero during a creation action, and one when the creation action has been committed.

This variable should **not** be used in the following properties:

- Map **Subject**, where it will always return a value of zero.
- Field **Variable**, since the variable is read-only.

%Archived

boolean **read-only** indication of whether a folder is archived. Returns `True` if the folder is or has ever been at an Archive stage, else returns `False`.

%Category

string **writeable** name (maximum 31 characters) used to categorize folders, under the control of the designer.

This variable is available in **all** properties.

%CreationTime

date-time **read-only** date and time at which the selected folder's creation action was invoked or (if already committed) committed.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%Deadline

date-time **writeable** date and time used to control timed actions that are based on the folder's deadline.

This variable is available in **all** properties.

%EntryTime

date-time **read-only** date and time at which the selected folder entered or re-entered its current stage. This is not reset on a loopback action.

This variable should **not** be used in the following properties:

- Map **Subject**, since during creation the folder is not at any stage.
- Field **Variable**, since the variable is read-only.

%FolderID

string **read-only** unique, system-generated, folder identifier.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%FolderName

string **writeable** user-readable folder name, which may or may not be unique. This variable is preset when the folder is created, from the map's ***Folder prefix*** and ***Length of numeric suffix*** properties, but may be changed during an action.

This variable is available in **all** properties.

%MapName

string **read-only** name of the map responsible for the folder.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%Message[event_number]

string **read-only** alert message generated for the specified ***event_number*** for this folder. The folder's creation is event number 0.

 © Metastorm Inc., 2008

This is evaluated via a database lookup, and thus may have an effect on performance. It should **not** be used in the following properties:

- Map **Subject**, since on creation there can be no earlier messages.
- Field **Variable**, since the variable is read-only.

%Notes[event_number]

string **read-only** value of **`%Action.Notes`** saved for the specified ***event_number*** for this folder. The folder's creation is event number 0.

This is evaluated via a database lookup, and thus may have an effect on performance. It should **not** be used in the following properties:

- Map **Subject**, since on creation there can be no earlier notes.
- Field **Variable**, since the variable is read-only.

%Originator

string **read-only** registered user who created this folder. This will be empty for a folder created by means of a flagged action.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%Parent

string **writeable** internal identifier of the folder's parent. This is preset automatically for folders created by means of a subprocedure, and for folders cloned from another folder, but may be changed during an action.

This variable is available in **all** properties.

%Priority

number **writeable** current priority (from zero to nine) of the folder.

This variable is available in **all** properties.

%ServerName

string **read-only**. When a folder is created, a new row is inserted into eFolder. The value used for the eServerName column is the cached server name, and this is the folder value reference by %ServerName.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%StageName

string **read-only** name of the folder's current stage. This will be empty during a creation action, and will identify the source stage of any other action.

This variable should **not** be used in the following property:

- Field **Variable**, since the variable is read-only.

%Subject

string **writeable** folder subject (maximum 250 characters). This variable is preset from the map's ***Subject*** property when the folder is created but may be changed during an action.

This variable is available in **all** properties.

%Updated

date-time **read-only** date and time at which the most recent action on the selected folder was committed.

This variable should **not** be used in the following property:

- Map **Subject**, since during creation the folder has not yet been updated.
- Field **Variable**, since the variable is read-only.

custom variables…

number **writeable** integer, real or currency variable.

string **writeable** text or memo variable.

list **writeable** memo variable.

date-time **writeable** date-time variable.

condition **writeable** check variable.

Custom variables are available in **all** properties.

Form Variables

%Field.[fieldname]

string **read-only**, giving the value of a field on a form. During the processing of a Metastorm event, such as **When changed**, it represents the value when event processing begins.

This variable should **only** be used in the following properties:

 © Metastorm Inc., 2008

- **When changed**.
- **When user saves form**.
- **When action completed**.

Pure Functions

Pure functions are those that return a result, but have no other effect on the system and require no access to the file system or database. They will always return the same value for any given set of arguments.

Pure functions may be used in any property, whether as part of an assignment *command*, or (more frequently) as stand-alone values used to return a value into a property. Pure functions may also be nested as the argument of another function call.

The following pure Metastorm functions are defined:

%Abs(value)

Arguments:

value	a *number* or *currency value* whose absolute (positive) value is required.

Returns:

Number	the absolute (non-negative) value of ***value***, irrespective of its sign.

If a currency value is passed in the result is also a currency value.

For example:

%Abs(-999.999) // 999.999

%Asc(character)

Arguments:

character	a *string* whose first character's ASCII code is required. Any characters after the first will be ignored.

Returns:

number	the ASCII code (0 to 255) for ***character*.**

For example:

%Asc("0") // 48

%Asc("012345") // 48 - Ignores rest of string

%Chr(code)

Arguments:

code	a *number*, in the range 0 to 255, whose ASCII character value is required.

Returns:

string	the single character represented by *code*.

For example:

%Chr(9)	// Horizontal tab
%Chr(32)	// " "
%Chr(44)	// ","
%Chr(48)	// "0"

%Concatenate(string { , string })

Arguments:

string	two or more *strings*, to be concatenated together.

Returns:

string	the concatenation of all specified ***strings***, with no intervening spaces.

For example:

%Concatenate("Bob",%Chr(32),"Smith")	// "Bob Smith"
%Concatenate(%Chr(13),%Chr(10))	// %NEWLINE()

 © Metastorm Inc., 2008

%Count(list ⟦ , delimiter ⟧)

Arguments:

list	a *list* or (if ***delimiter*** specified) a *string* whose entries are to be counted.
delimiter	an *optional string* delimiter between successive entries in ***list***. If omitted or empty, a comma is used.

Returns:

number	the number of elements in ***list***, where each element is separated by ***delimiter***. Empty elements are *counted*. An empty list will return a count of zero. An empty list containing only delimiters will return one more than the number of delimiters.

For example:

%Count("Alpha,Beta,Gamma") **// 3**

%Count(%Concatenate("Line 1",%NEWLINE(),"Line 2"),%NEWLINE()) **// 2**

%Count("C++","+") **// 3**

% Duration (from, to ⟦ , units ⟧)

Arguments:

from	a *date-time* from which a duration should start.
to	a *date-time* at which the duration should end.
units	an optional *string* specifying the units of duration, as follows: **minutes (or** mi**)** **hours (or** h**)** **days (or** d**)** **weekdays (or** wys**)** **weeks (or** wks**)** **months (or** mo**)** **years (or** y**)** If this is omitted or empty, it will default to days.

Returns:

number	a real number. The number of *units* between *from* and *to*. If the end time precedes the start time, then this will be a negative value.

For example:

%Duration(%Deadline,%System.Time,"weekdays") **// Weekdays since deadline**

%Duration(%CreationTime,%system.Time,"hours") **// Age of folder, in hours**

%EMPTY()

Returns:

string	an empty string. This may be used to clear string and list variables.

For example:

```
%Subject := %EMPTY()          // Clear %Subject
```

%Find(substring, instring, case ⟦ , start_position ⟧)

Arguments:

substring	a *string* to be searched for.
instring	a *string* to search through.
case	a *number* indicating whether the search should (case 1) or should not (case 0) be case sensitive.
start_position	an optional number indicating the position within ***instring*** after which the search should start. The first character position within any string is 1, so to search from the start of the string this value should be set to 1. If omitted or empty, it defaults to 1.

Returns:

number	The starting position (from 1) of ***substring*** within ***instring***, taking ***case*** and ***start_position*** into account if necessary. Returns 0 if the substring cannot be found.

For example:

```
%Find("abc","ABCabcabc",0)          // 1
%Find("abc","ABCabcabc",1)          // 4
%Find("abc","ABCabcabc",0,4)        // 7
```

%FindEntry(list, item ⟦ , delimiter ⟧)

Arguments:

list	a *list* or (if ***delimiter*** specified) a *string* whose entries are to be searched.
item	a *string* entry to be found within ***list***.
delimiter	an optional *string* delimiter between successive entries in ***list***. If omitted or empty, a comma is used as the delimiter.

Returns:

number	the position of ***item*** within ***list***, using ***delimiter***. The list is zero-based: the first item in the list is item 0. Returns -1 if the item is not in the list.

For example:

```
%FindEntry("a,b,c","b")      // 1
%FindEntry("a,b,c","d")      // -1
%FindEntry(%Session.FlagData,%myCustomerName,%Chr(9))
```

%FormatCurrency(currency-value, ⟦ locale, ⟦ digits, leading, group, decimal-separator, thousand-separator, negative-order, positive-order, currency-symbol⟧ ⟧)

Arguments:

currency-value	a numeric *string* to be formatted as a currency.
locale	an *integer* that determines the format for the entire result, and for symbols such as the negative number indicator. If the locale number is not known, the following strings may be used: ▪ DEU for German ▪ ENG for U.K. English ▪ ENU for U.S. English ▪ ESN for Modern Spanish ▪ ESP for Castilian Spanish ▪ FRA for French ▪ FRC for Canadian French If the following arguments are not included, this argument is optional. If it is not provided, the LOCALE_SYSTEM_DEFAULT is used.

digits an optional *integer* that indicates the number of digits to pad after the decimal point.

leading an optional *Boolean.* If TRUE, a leading digit will be included.

group an optional *integer* that indicates the size of each group of digits to the left of the decimal. The valid range is 0 – 9.

decimal-separator an optional *string* that will be used as the decimal separator.

thousand-separator an optional *string* that will be used as the thousand separator.

negative-order an optional *integer* that indicates the format for a negative currency value. The following values are supported:

- 0 ($1.1)
- 1 -$1.1
- 2 $-1.1
- 3 $1.1-
- 4 (1.1$)
- 5 -1.1$
- 6 1.1-$
- 7 1.1$-
- 8 -1.1 $
- 9 -$ 1.1
- 10 1.1 $-
- 11 $ 1.1-
- 12 $ -1.1
- 13 1.1 -$
- 14 ($ 1.1)
- 15 (1.1 $)

positive-order an optional *integer* that indicates the position of the monetary symbol in a positive currency value. The following values are supported:

- 0 Prefix, no separation, e.g. $1.1
- 1 Suffix, no separation, e.g. 1.1$
- 2 Prefix, 1 character separation, e.g. $ 1.1
- 3 Suffix, 1 character separation, e.g. 1.1 $

currency-symbol an optional *string* that will be used as the currency symbol.

Returns:

string a representation of ***currency***, formatted as specified by the ***locale*** and other parameters.

For example:

%FormatCurrency(14.50) // **"$14.50"**

%FormatCurrency(14.50, ENU) // **"$14.50"**

%FormatCurrency(146000.50, ENU, 1, TRUE, 3,".",",",1,0) // **"$146,000.5"**

%FormatNumber(number, decimals ⟦ , width ⟦ , format ⟧ ⟧)

Arguments:

number	a *number* to be formatted.
decimals	the *number* of decimal places required in the result.
width	an optional minimum *number* of characters required in the result, which will be padded as necessary. If the formatted number is longer than this, then the result will be extended accordingly. If omitted or empty, a minimum width of 0 will be used.
format	an optional *string* used to specify the format, built from the following elements:

0	**pad with leading zeros (rather than spaces)**
-	**align to left (rather than right) within width**
+	**prefix positive numbers with explicit + sign**
p	**prefix positive numbers with a space character**
#	**always display decimal portion, even if zero**
b	**return empty string if value is zero**
%%	**return as percentage with trailing % character**

Returns:

string	a representation of ***number***, formatted as specified by ***decimals***, ***width*** and ***format***.

For example:

%FormatNumber(12.34,0) // **"12"**

%FormatNumber(12.34,1) // **"12.3"**

%FormatNumber(12.34,1,5) // **" 12.3"**

%FormatNumber(12.34,2,,"+") // **"+12.34"**

%FormatNumber(12.34,2,7,"0+") // **"+012.34"**

%FormatNumber(12.34,0,4,"%%") // **" 12%"**

%FormatTime(timestamp, format)

Arguments:

timestamp	a *date-time* (service time) to be formatted.

format a *string* used to specify the format, built from the following elements:

c	**general date format (from server locale)**
dddddd	**long date format (from server locale)**
ddddd	**short date format (from server locale)**
dddd	**full name of weekday (from server locale)**
ddd	**first three letters of weekday name**
dd	**two-digit day of month (01 to 31)**
d	**one- or two-digit day of month (1 to 31)**
ww	**one- or two-digit week of year (0 to 52)**
w	**one-digit day of week (0 to 6, Sunday as 0)**
mmmm	**full name of month (from server locale)**
mmm	**first three letters of month name**
mm	**two-digit month of year (01 to 12)**
m	**one- or two-digit month of year (1 to 12)**
yyyy	**four-digit year (0000 to 9999)**
yy	**two-digit year (00 to 99)**
y	**one- to three-digit day of year (1 to 366)**
hhh	**one- or two-digit hour o'clock (1 to 12)**
hh	**two-digit hour of day (00 to 23)**
h	**one- or two-digit hour of day (0 to 23)**
nn	**two-digit minute of hour (00 to 59)**
n	**one- or two-digit minute of hour (0 to 59)**
ss	**two-digit second of minute (00 to 59)**
s	**one- or two-digit second of minute (0 to 59)**
tttt	**long time format (from server locale)**
AM/PM	**hh:nn:ss with AM** or **PM indicator**

Returns:

string a representation of ***timestamp***, formatted as specified

For example:

```
%FormatTime(%System.Time,yyyymmddhhnn)      // "200011281830"
```

 © Metastorm Inc., 2008

%GetEntry(list, index [, delimiter])

Arguments:

list	a *list* or (if ***delimiter*** specified) a *string* from which an entry is to be returned.
index	a (zero-based) *number* index of the entry to be returned.
delimiter	an optional *string* delimiter between successive entries in list. If omitted, a comma is assumed to be the delimiter.

Returns:

string	a copy of entry number ***index*** from ***list***.

For example:

```
%GetEntry("a,b,c",0)          // "a"
%GetEntry(%ToDoList(%FolderID),%myIndex)
%GetEntry(%Session.FlagData,1,%Chr(9))
```

%Length(string)

Arguments:

string	a *string* whose length (total character count) is required.

Returns:

number	the number of characters in ***string***.

For example:

```
%Length("abcdef")             // 6
```

%LINEFEED()

Returns:

string	a line feed character (ASCII **<LF>**; decimal **<10>**). This is used as a line delimiter in *strings*.

For example:

```
%Concatenate(%myMemo1,%LINEFEED(),%myMemo2)
```

In Version 6, a new XML-based Transaction Protocol (TP) was introduced. The XML Parser, which is used by the Process Engine, ensures that the TP XML is normalized. Part of the XML normalization process results in all line breaks being represented by a Line Feed character. Until the XML is normalized, a combination of Carriage Return and Line Feed characters may represent line breaks.

In general, all TP requests and responses passing through the Process Engine are normalized. However, there are some exceptions. The following list summarizes normalization of the TP XML in the Engine:

- *Any data that comes in as a TP request is normalized as it enters the Engine.*
- *Any data that is returned from the Engine in a TP response is normalized before it leaves the Engine.*
- *Assignments to custom variables are not normalized for the duration of the request in which the assignment was made. Any processing carried out by the Engine will take place on the current values. However, a field that displays a custom variable will in effect normalize that custom variable value when it is passed back in the next TP request.*

For example, if When Stage Started sets a custom variable to %NewLine, the value stored for this custom variable in the database is 'CRLF'. This remains the case until a form containing a memo field displaying this custom variable is displayed to the user. As the custom variable has been sent to the Client via the TP in XML format, its value has been normalized and the value displayed on the form is 'LF'. When the form is submitted, the value sent back to the Engine for this custom variable is 'LF'. The value stored in the database is now 'LF'.

Process designers should be aware of this normalization process and ensure that any calculation formulas that reference a %NewLine (which returns a Carriage Return and Line Feed pair) are updated to reference a %LineFeed (which just returns a Line Feed character) where necessary.

%MakeDate(count, units, direction, from)

Arguments:

count	a *number* of time ***units*** to count.
units	a *string* specifying the units of duration, as follows: **minutes (or** mi**)** **hours (or** h**)** **days (or** d**)** **weekdays (or** wys**)** **weeks (or** wks**)** **months (or** mo**)** **years (or** y**)** If omitted or empty, it defaults to days.
direction	a *string* specifying the direction in which to count, as follows: **before (or** b**)** **after (or** a**)**
from	a *date-time* from which to count.

Returns:

date-time	the date and time which is the specified ***count*** of time ***units*** before or after (***direction***) the date and time ***from***.

For example:

```
%MakeDate(1,days,after,%System.Time)      // Same time tomorrow
```

%Max(value1 , [value2])

Arguments:

value one of a list of numbers or currency values whose maximum value is required. One or more values may be passed in.

Returns:

number or *currency* the highest numeric or currency value in the list. If all the values that are passed in are currencies (or cast as currencies) the highest currency value is calculated and returned, else the highest numeric value is returned.

For example:

%Max(%CurrencyValue1, %CurrencyValue2) // **highest currency**

%Max(%NumberValue1, %NumberValue2) // **highest number**

%Max(%CurrencyValue1, (currency)%NumberValue1) // **highest currency**

%Max(%CurrencyValue1, %NumberValue1) // **highest number**

%Max(9.99,2.50,5.95) // **9.99**

%Max(list)

Arguments:

list a *list* of numbers whose maximum value is required.

Returns:

number the highest numeric value in ***list***.

For example:

%Max("9.9,9999,99.99,-999999") // **9999**

%Max("%myNumber1,%myNumber2")

%Max(%myNumberList)

%Min(value1 , [value2])

Arguments:

Value one of a list of numbers or currency values whose minimum value is required. One or more values may be passed in.

Returns:

number or *currency* the lowest numeric or currency value in the list. If all the values that are passed in are currencies (or cast as currencies) the lowest currency value is calculated and returned, else the lowest numeric value is returned.

For example:

%Min(%CurrencyValue1, %CurrencyValue2) // **lowest currency**

%Min(%NumberValue1, %NumberValue2) // **lowest number**

%Min(%CurrencyValue1, (currency)%NumberValue1) // **lowest currency**

%Min(%CurrencyValue1, %NumberValue1) // **lowest number**

%Min(9.99,2.50,5.95) // **2.50**

%Min(list)

Arguments:

list a *list* of numbers whose minimum value is required.

Returns:

number the lowest numeric value in ***list***.

For example:

%Min("9.9,9999,99.99,-999999") // **-999999**

%Min("%myNumber1,%myNumber2")

%Min(%myNumberList)

%Months(start-date, months)

Arguments:

start-date the *date* from which to calculate

months an *integer* representing the number of months to add

Returns:

date the resulting *date* after the number of months specified by **months** has been added to the **start-date**

For example:

%Months("2/9/2004",1) **// Returns "3/9/2004"**

%NEVER()

Returns:

date-time a null (unspecified) date and time.

 © Metastorm Inc., 2008

For example:

%Deadline := %NEVER() // **Clear the folder deadline**

%NEWLINE()

Returns:

string	a carriage return, line feed pair (ASCII **<CR><LF>**; decimal **<13><10>**). This is used as a line delimiter in *strings* and between successive rows of a database table.

For example:

%Concatenate(%myMemo1,%NEWLINE(),%myMemo2)

☞ *For further information, refer to the description of the %LineFeed function.*

%Rem(dividend, divisor)

Arguments:

dividend	a *number* to be divided.
divisor	a *number* to divide by.

Returns:

number	the integer remainder from the division of ***dividend*** by ***divisor***.

For example:

%Rem(50,7) // **1**

%RemoveEntry(list, index ⟦ , delimiter ⟧)

Arguments:

list	a *list* or (if ***delimiter*** specified) *string* from which an entry is to be removed.
index	a whole (integer) *number* index (zero-based) of the entry to be removed.
delimiter	an optional *string* delimiter between successive entries in list. If omitted, a comma is assumed to be the delimiter.

Returns:

list	a copy of ***list***, with entry number ***index*** removed.

For example:

%RemoveEntry("a,b,c",0) // **"b,c"**

%myList := %RemoveEntry(%myList,%myIndex)

%Replace(source, old, new ⟦ , count ⟧)

Arguments:

source	a *string* in which the replacement is to be made.
old	a *string* to be replaced.
new	a *string* to replace ***old*** by.
count	an optional number of occurrences of ***old*** to be replaced. If omitted (or 0), all occurrences will be replaced.

Returns:

string	**a new version of** *source***, with all** replacements **made.**

For example:

%Replace("abra","a","A") // **"AbrA"**

%Replace("abra","a","A",1) // **"Abra"**

%Replace(%myMemo,%NEWLINE(),%Chr(44)) // **multiple lines => csv**

%Replace(%myCSV,%Chr(44),%Chr(9)) // **csv => tab-separated**

%Replace(%myTSV,%Chr(9),%NEWLINE()) // **tsv => multiple lines**

%Round(number ⟦ , decimals ⟦ , round-down ⟧ ⟧)

Arguments:

number	a *number* or *currency value* to be rounded.
decimals	an optional *number* of decimal places to be rounded to. If omitted or empty, number will be rounded to a whole number.
round-down	an optional *number* indicating whether a value of 0.5 or greater should be rounded up (0) or down (1). If omitted or empty, number will be rounded up.

Returns:

number	**the rounded value.**

If a currency value was passed in, the result is a currency value. This function supports 'Bankers Rounding' for currencies, and 'Engineers Rounding' for Real numbers.

For example:

%Round(99.995) // **100**

%Round(99.995,2) // **100.00**

%Round(99.995,2,1) // **99.99**

%Round(1030,-3) // **1000**

%Substring(start, length, string)

Arguments:

start	a *number* identifying the first character to be copied (where the first character in the string is character 1).
length	a *number* of characters to be copied.
string	a *string* from which a substring is to be copied.

Returns:

string	**the copied** substring**.**

For example:

%Substring(3,3,"abcdef")	**// "cde"**
%Substring(4,3,"ABCDEF")	**// "DEF"**

%Weekdays(start-date, days)

Arguments:

start-date	the *date* from which to calculate
days	an *integer* representing the number of days to add

Returns:

date	the resulting *date* after the number of days specified by **days** has been added to the **start-date**

For example:

%Weekdays(“2/8/2004”,1)	**// Returns “2/9/2004”**

Access Functions

Access functions return a result, obtained via access to the file system or database, but have no other effect on the system. They will generally return different values for any given set of arguments, depending on the state of the file system and database when they are called.

Access functions may be used in any property, whether as part of an assignment *command*, or (more frequently) as stand-alone values used to return a value into a property. Access functions may also be nested as the argument of another function call.

The following access functions are defined:

%CaptureAttachment (folderID, filename, attachmentData)

This function allows a user to add a new attachment to an existing folder. It can also be used to assign that folder to a clip field. For this to work, the user should assign the return value of this function to the clip field:

```
%clipField := %CaptureAttachment( %folderID, “test.txt”,
%Action.Notes[1] )
```

Remember the value returned may not be the same as the filename requested. This is because a given folder can only contain one attachment with a given name.

Arguments:

folderID	a *string* containing the Folder ID for the attachment.
filename	a *string* containing the filename the Metastorm Engine attempts to give the attachment.
attachmentData	a *string* containing the attachment's contents. For binary files, this should be Base64 encoded.

Returns:

string	unique name given to the attachment by Metastorm BPM.

%Dir(path, [Alt Delimiter])

Arguments:

path	a *string* identifying the (server) directory path whose files are to be listed. Note that sub-directories are not listed.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

a *string* containing the directory listing for the specified ***path***, obtained from the server file system, with one file per line, and the following tab-delimited values for each file:

File name.extension

File size (in Kbytes)

Modification date (server locale format)

 © Metastorm Inc., 2008

For example:

```
%Dir("C:\My Documents")
```

%EmailAddress(user, [Alt Delimiter])

Arguments:

user	a *string* identifying the registered Metastorm user whose email address is required.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

string	the email address of the specified ***user***, obtained from the Metastorm database.

For example:

```
%EmailAddress(%User.Name)                    // Email address of current user
%EmailAddress(%Manager(%Originator,VP,CEO))  // Of originator's VP, or of CEO
```

%GetData(data_source, table, rows, column, [col_delimiter], [row_delimiter])

Arguments:

data_source	a connection *string* for the database containing the information to be listed. If empty, the Metastorm database will be used.

You should not use a DSN string, when mapping to the Metastorm database, as it could cause database locks.

table	a *string* identifying the table containing the information to be listed.
rows	a *string* to be used in a SQL **`WHERE`** clause to select the required rows from ***table***. If empty, all (distinct) rows will be returned.
column	a *string* identifying the column whose contents are to be listed.
col_delimiter	an optional *string* delimiter to be placed between each column in the result set. If omitted or empty, a comma is used as the delimiter.
row_delimiter	an optional *string* delimiter to be placed between each row in the result set. If omitted or empty, a comma is used as the delimiter.

Returns:

list	an alphabetically sorted list of unique values from the specified ***rows*** and ***column*** of the specified ***table*** in the specified (or Metastorm) ***data_source***.

For example:

```
%GetData(,eFolder,"eParent='%FolderID'",eFolderName)      // All child folder names

%GetData("DSN=CustData;UID=sa;PWD=xyzzy",
  CustomerTable,"CountryCode='USA'",CustomerName)         // All US customers
```

%GetFolder(table, folder, variable, [Alt Delimiter])

Arguments:

table	a *string* identifying the custom table holding the required custom variable.
folder	a *string* to be used in a SQL **`WHERE`** clause to select the required row from ***table***.
variable	a *string* identifying the custom variable column whose value is required.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

string	the value of the custom ***variable***, for the specified ***folder***, held in the specified map ***table*** in the Metastorm database.

For example:

```
%GetFolder(Flight,"eFolder.eFolderID='%Parent'",Class)
                                          // Parent folder's Class variable
```

%GetText(filename)

Arguments:

filename	a *string* identifying the (server) directory path, name and extension of a text file whose contents are to be read.

Returns:

string	the contents of the specified ***filename***, obtained from the server file system.

For example:

```
%GetText("C:\My Documents\Text file.txt")
```

This function supports both ANSI and Unicode files. The function examines the first two bytes of the file it is reading; if it detects byte-order mark it treats the file it is reading as Unicode, otherwise it assumes that the file is ANSI.

 © Metastorm Inc., 2008

%Manager(userID, role [, fallback_role])

Arguments:

userID	a string identifying the registered Metastorm user whose manager is required.
role	a *string* identifying a role that the manager must hold.
fallback_role	a *string* identifying a role to be returned in the event that ***userID*** has no ***manager*** with the specified ***role***.

Returns:

string	the ID of the registered Metastorm user to whom ***userID*** reports (or to whom the person user reports to reports, and so on), who also holds ***role***. If no such user can be found, an empty string or the optional ***fallback_role*** will be returned. This information is obtained from the Metastorm database.

For example:

```
%Manager(%User.Name,everybody,%User.Name)   // User's direct manager (or user)
%Manager(%Originator,VP,CEO)                // Originator's VP (or CEO)
```

%ReadAttachment (folderID, filename)

Arguments:

folderID	a *string* value containing the Folder ID for the attachment.
filename	a *string* value containing the name of the attachment.

Returns:

string	the Base64-encoded attachment.

%ScriptEval(language, file, procedure, map, expression [, args])

Arguments:

language	a *string* identifying the scripting language processor to be loaded (VBScript, JScript and JScript.NET are supported by the Designer).
file	if the specified ***language*** is VBScript or JScript, this is a *string* identifying the (server) directory path, name and extension of a script file to be loaded. If empty, no script will be loaded from file. If the specified ***language*** is JScript .NET, this *string* is a list of files to be added to the script host when the assembly is compiled. These files will only be looked at when the script is compiled. Compilation occurs the first time it is used, on Engine start-up, and for every new version published thereafter.
procedure	if the specified ***language*** is VBScript or JScript, this is a *string* identifying a procedure holding a script attachment to be loaded. If empty, no procedure script will be loaded. If the specified ***language*** is JScript .NET, this *string* is the name of the procedure to be pulled in when the assembly is compiled. This can be left blank to indicate no procedure script.
map	if the specified ***language*** is VBScript or JScript, this is a *string* identifying a map holding a script attachment to be loaded. If empty, no map script will be loaded. If the specified ***language*** is JScript .NET, this *string* is the name of the map script to be pulled in when the assembly is compiled. This can be left blank to indicate no map script.
expression	if the specified ***language*** is VBScript or JScript, this is a *string* containing an expression, in the specified ***language***, to be evaluated. This will typically be a call to a function within one or other of the scripts loaded from the specified ***file***, ***procedure*** and/or ***map***. If any arguments need to be **"**quoted**"** (as string arguments do for VBScript), then each quotation mark must be preceded by a `%` character to prevent Metastorm BPM 'stealing' it before it is passed to the script function. If the specified ***language*** is JScript .NET, this a *string* indicating the name of a valid entry point in the map or procedure assembly that has been compiled.
arg	(JScript .NET only) any number of additional arguments to be packaged up and passed into the entry point's `args` argument.

Returns:

string	if the specified ***language*** is VBScript or JScript, this returns the result of the evaluation. The function loads the specified scripting ***language***, followed by any script in ***file*** (from the server file system), any specified ***procedure*** and/or ***map*** scripts

 © Metastorm Inc., 2008

(from the Metastorm database), and then evaluates the specified ***expression***.

If the specified ***language*** is JScript .NET, this is the value returned by the invoked entry point. If the entry point returns nothing, this will return True.

The evaluation suite returns the result to the Engine once the expression has been evaluated.

Note that if the script executed as a result of this function call has side effects on the system, this becomes an impure function, which should only be used in the following properties:

- Stage When stage completed.
- Stage When stage started.
- Action When action completed.
- Form When user saves form.
- Button field When button pressed.
- Grid field When user selects row.

For example:

```
%FolderName := %ScriptEval(VBScript,"",
        %SelectSQL("SELECT eProcedureName FROM eMap
              WHERE ework.eMapName='%MapName'"),,
                  "NewFolderName(%"%MapName%")")     // Returns new folder name
```

%SelectSQL(statement ⟦ , data-source ⟦ , col_delimiter ⟦ , row_delimiter ⟦ , headers ⟧ ⟧ ⟧ ⟧)

Arguments:

statement	a *string* containing the SQL statement to be executed. The statement must be a **`SELECT`** (use **`%ExecSQL`** instead, if you do not need to return a result set).
data-source	an optional connection *string* for the database against which the SQL statement is to be executed. If omitted or empty, the SQL will be executed against the Metastorm database.

You should not use a DSN string, when mapping to the Metastorm database, as it could cause database locks.

col_delimiter	an optional *string* delimiter to be placed between each column in the result set. If omitted or empty, a horizontal tab character is used as the delimiter.
row_delimiter	an optional *string* delimiter to be placed between each row in the result set. If omitted or empty, a new line (carriage return / line feed pair) is used as the delimiter.
headers	an optional *Boolean*. If this is set to TRUE, the first row of the result set will be the column headers. If omitted, this defaults to FALSE.

If you request headers and the SELECT produces no output, the headers will still appear.

Returns:

string	the requested result set, with the relevant column and row delimiters, obtained from the specified (or Metastorm) ***data_source***.

For example:

```
%SelectSQL("SELECT * FROM eUser WHERE eUserName = '%User.Name'")
                                                    // Return user details

%SelectSQL(                                         // Read external data
      "SELECT * FROM CustomerTable
            WHERE CustomerName = '%myCustomerName'",
      "DSN=CustomerData;UID=sa;PWD=")


%SelectSQL(                                         // Format folder history
      "SELECT eEventTime,' – ',eUserName,' – ',eAlertMessage,
            '%NEWLINE()',eNotes
            FROM eEvent
            WHERE eFolderID='%FolderID' AND eNotes IS NOT NULL
            ORDER BY eEventID",
      "",                                           // From Metastorm database
      "",                                           // No column separators
      "%NEWLINE()%NEWLINE()")                       // Two new lines between rows
```

Using the DISTINCT or UNIQUE SQL keywords in Metastorm formulas running against some Oracle servers may cause an error in the formula evaluation. For example, the following valid queries may fail:

SELECT COUNT(DISTINCT eUserName) FROM eUser
SELECT COUNT(UNIQUE eUserName) FROM eUser

To ensure these queries run with Metastorm BPM, modify them as follows:
SELECT COUNT(DISTINCT (eUserName)) FROM eUser
SELECT COUNT(UNIQUE (eUserName)) FROM eUser

 © Metastorm Inc., 2008

%Staff(userID, role , fallback-role, [Alt Delimiter])

Arguments:

userID	a *string* identifying the registered Metastorm user whose staff are required.
role	a *string* identifying a role that the staff must hold.
fallback_role	a *string* identifying a role to be returned in the event that ***userID*** has no staff with the specified ***role***.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

list	a comma-delimited list of staff members who hold ***role*** and directly report to the ***userID.***. If no such staff members can be found, an empty string or the staff holding the mandatory ***fall-back role*** will be returned. This information is obtained from the Metastorm database.

For example:

```
%Staff(%User.Name,everybody,"")          // All user's immediate reports
%Staff(%Originator,"secretary",%Originator)// User's secretary, or self
```

%ToDoList(folderID, [Alt Delimiter])

Arguments:

folderID	a *string* identifying the folder which the users to be listed have on their ToDo lists.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

list	the list of all users who have the specified folder on their ToDo list, obtained from the Metastorm database.

For example:

```
%ToDoList(%FolderID)                     // All users with this To Do
```

%WatchList(folderID, [Alt Delimiter])

Arguments:

folderID	a *string* identifying the folder which the users to be listed have on their Watch lists.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

list	the list of all users who have the specified folder on their Watch list, obtained from the Metastorm database.

For example:

%WatchList(%Parent) // **All users watching parent folder**

%LDAPSearch(directory_alias, filter_expression, attribute_name, [search_scope], [Alt Delimiter])

Arguments:

directory_alias	the name of an LDAP alias that contains connection details for the required directory.
filter_expression	an LDAP filter expression which specifies which directory entries should be returned. For more information about filter expressions, refer to section 10.1.7, Accessing Directories (Direct Directory Access).
attribute_name	the name of the attribute to query from each directory entry matching the search criteria. You may retrieve one attribute per %LDAPSearch function. In addition to attributes defined in the directory schema, you may request attributes created using the Metastorm snap-ins for Microsoft Active Directory and Novell eDirectory. To request a custom attribute, use the format "ework:<attribute name>", where <attribute name> is the name of the custom attribute.
search_scope	this parameter is not currently used.
Alt Delimiter	an optional *string* specifying the delimiter to use in place of a comma

Returns:

text	information for the specified attribute , as a result of the filter expression applied to the specified directory.

For example:

```
%LDAPSearch(Directory1,"(objectClass=inetOrgPerson)","cn",) //
```
Return the cn attribute value from every inetOrgPerson entry in the location specified by the Directory1 alias.

Impure Functions

Impure functions are those that have some side effect, such as writing to a file or database or sending an email. They may or may not return a result: throughout this section, those that do not return a result are shown as *commands*, and those that do return a result are shown as part of an assignment *command*.

Impure functions should only be used in the following properties:

- Stage When stage completed.
- Stage When stage started.

- Action When action completed.
- Form When user saves form.
- Button field When button pressed.
- Grid field When user selects row.

The following impure Metastorm functions are defined:

%Delete(filename)

Arguments:

filename	a *string* identifying the (server) directory path, name and extension to be deleted.

Returns:

condition	**TRUE** if ***filename*** was deleted from the server file system, **FALSE** otherwise.

For example:

```
%Delete("C:\Windows\Temp\Temporary file.tmp")
    ?(%User.Error := %EMPTY())
    :(%User.Error := "File could not be deleted")
```

%DeleteAttachment(folderID, filename)

Arguments:

folderID	a *string* identifying the folder whose attachment is to be deleted.
filename	a *string* containing the file name and extension (without path) of an attachment to be deleted from the folder.

Returns:

string	an empty string, used to reset the value of any clip variable associated with the attachment after deleting the attachment from the Metastorm database.

For example:

```
%myClip := %DeleteAttachment(%FolderID,%myClip)  // Clears %myClip
```

The %DeleteAttachment function removes the specified attachment file from the database. It does not however reset the value of the custom variable that is associated with a clip. As a result, after invoking %DeleteAttachment, a clip associated with the deleted attachment will appear as if the attachment still exists. The work-around for this problem is to reset the custom variable after invoking %DeleteAttachment as follows:

```
%DeleteAttachment(%folderID,"%txtAttachmentPath")
%txtAttachmentPath := ""
```

%EMail(toList, ccList, subject, message ⟦ , attachment ⟦ , reply address ⟧ ⟧)

This function makes use of Microsoft's cdonts.dll. which is not officially supported on any version of Windows after Windows 2000 (although it appears to work). For this reason, it is recommended that the 'Send an e-mail message' function (provided by the Mail library) is used instead of %EMail.

Arguments:

tolist	a *string* containing the To list for an email.
cclist	a *string* containing the CC list for the email.
subject	a *string* containing the subject for the email.
message	a *string* containing the body of the email.
attachment	an optional *string* identifying the (server) directory path, name and extension of a file to be included as an attachment. If omitted or empty, no attachment will be included.
reply address	an optional *string* containing the email address of the person who sent the message. This will only be valuable if the mail system in use supports return addresses. If the *reply address* argument is missing then the engine's default email address is used. This is only true for a mail connector that supports the *reply address* argument (SMTP does, MAPI doesn't).

Returns:

command	Sends an email to the specified users, copying the specified ***cc*** list, with the specified subject, ***message***, ***attachment*** and ***reply address***. The function executes asynchronously: the Engine continues processing as soon as the request has been queued, not after it has been serviced.

For example:

```
%EMail("helpdesk@metastorm.com","",%Subject,%Action.Notes)
```

%ExecProc(procedure, arguments)

The %ExecProc function is similar to the %Execute function.

The Money data type is not supported with the %Execute function. To run a stored procedure that uses a Money data type, use the %ExecProc function.

 © Metastorm Inc., 2008

Arguments:

procedure	is a *string* identifying a database stored procedure.
arguments	is one or more arguments of any data type to be supplied to the stored procedure. The values are passed to the stored procedure in their current type. So: %ExecProc("procedure1", %CurrencyVariable, %RealVariable, %TextVariable, %DateVariable) will pass 4 arguments of types currency, real, text and date.

Returns:

command	executes the specified stored ***procedure***, with any specified ***arguments***, against the Metastorm database.

%ExecProc does not support output parameters.

If you pass in a parameter of type CLOB, the value of this parameter must not be an empty string.

%ExecSQL(statement ⟦ , data_source ⟧)

Arguments:

statement	a *string* containing the SQL statement to be executed. The statement should not return a result set (use **`%SelectSQL`** instead, if you need to return a result set).
data_source	an optional connection *string* for the database against which the SQL statement is to be executed. If omitted, the SQL will be executed against the Metastorm database.

You should not use a DSN string, when mapping to the Metastorm database, as it could cause database locks.

Returns:

command	executes the specified SQL ***statement*** against either the Metastorm database or the specified ***data_source***.

For example:

```
%ExecSQL(
    "UPDATE CustomerTable
        SET CustomerName = '%myNewCustomerName'
        WHERE CustomerName = '%myOldCustomerName'",
"DSN=CustomerData;UID=sa;PWD=")
```

%Execute(procedure, arguments)

Arguments:

procedure	is a *string* identifying a database stored procedure.
arguments	is either a *string* of arguments or more than one *string* argument to be supplied to the stored procedure.

Returns:

command	executes the specified stored ***procedure***, with any specified ***arguments***, against the Metastorm database.

%Execute does not support output parameters.

For example:

This function can be used with two parameters (for backward compatibility) or multiple parameters. The **arguments** parameter may be one string containing a list of arguments, or a number of strings, each of which is an argument. If the stored procedure is does not accept any arguments, the **argument** parameter should not be included.

Two parameters:

```
%Execute(myProc,"arg1, arg2, arg3")          // Execute with arguments
```

Multiple parameters:

```
%Execute(myProc,"arg1","arg2","arg3")        // Execute with arguments
```

No arguments:

```
%Execute(myProc)                             // Execute without arguments
```

The Money data type is not supported with the %Execute function. To run a stored procedure that uses a Money data type, use the %ExecProc function.

*The %Execute function no longer supports the following format for the **argument** parameter: "'arg1','arg2','arg3'".*

If you pass in a parameter of type CLOB, the value of this parameter must not be an empty string.

%ExecuteExtensionEval (JScript.NET, %Procedure.Name, %MapName, "Metastorm.ProcessEvents.HandleExternalEvents.DelegateEvents" ,)

The above script is created when using the Integration Wizard and selecting the category **Event Handlers** and item **External Event Handler**.

To use this function, you must publish the Process Events Library and associate the library with the current procedure.

The function identifies the procedure and map names and calls the relevant stage, action or form event from Visual Studio .NET assemblies.

`(boolean)%ExecuteExtensionEval(JScript.NET,%Procedure.Name,%MapName, "Metastorm.ProcessEvents.HandleExternalEvents.DelegateEvents",)` is used when External Event Handler (Boolean) is selected from the Integration Wizard. This script is used for a conditional action which returns true or false.

For further information on creating custom extensions refer to Process Orchestrator for .NET Designer's Guide.pdf

%ExecuteRule(ruleConnection,customVariables)

To use this function, you must publish the BRI library.

You can use the %ExecuteRule function in a standard **'When…'** event or the **Rules** property of a Rules stage.

Arguments:

ruleConnection	A *String* specifying the rule to execute and the connection information required by the business rules engine.

Any changes to the browser proxy settings, while the Designer is open, require the Designer to be restarted.

customVariables	Optional list of *String* values specifying custom variables to pass to the rule. If none are specified, all custom variables are passed to the rule. Note that all system variables are always available to the rule.

Returns:

string	Result returned by the business rule.

For example:

%ExecuteRule("Alias=MyRule; ruleSetName=Rule1; URL=http://RuleEngineWebService; AuthDomain=; AuthMethod=1; BypassProxyOnLocal=True", Var2, Var3)

If the Metastorm Rules web service is on an IIS machine set up to use SSL, the machine calling the web service must have the certificate for the web service machine's web server in its trusted root certification authorities. It also needs a certificate from the certification authority.

%GetAttachment(folderID, filename, path)

Arguments:

folderID	a *string* identifying the folder whose attachment is to be saved to a file.
filename	a *string* ***containing*** the file name and extension (without path) of an attachment to be saved to file.
path	a *string* identifying ***the*** (server) directory path to which ***filename*** should be saved.

Returns:

command	extracts the specified ***filename*** attachment from the specified ***folderID*** in the Metastorm ***database***, and saves it to the specified path in the server file system.

For example:

```
%GetAttachment(%FolderID,%myClip,"C:\TEMP")
```

%GetAttachment does not accept unicode filenames.

%NewAttachment(folderID, filename, path)

Arguments:

folderID	a *string* identifying the folder to ***which*** an attachment should be made.
filename	a *string* containing the file name and extension (without path) of a file to be attached.
path	a *string* identifying the (server) ***directory*** path from which ***filename*** should be read.

Returns:

string	the attachment name and extension assigned to a new attachment created in the Metastorm database. The return value will differ from ***filename*** if an attachment with the specified name and extension already exists in the database for ***folderID***. This return value is typically assigned to a clip variable.

For example:

```
%myClip := %NewAttachment(%FolderID,
"Initial Application Form.doc","C:\Temp")
```

 © Metastorm Inc., 2008

%Print(document, folderID, printer, filename)

Arguments:

document	a *string* identifying the (server) directory path, name and extension of the MS Word document to be printed. *To perform a mail merge as efficiently as possible, ensure the template is read-only.*
folderID	the *string* **`%FolderID`**. If empty, error message displays, "Unable to Commit Action."
printer	a *string* specifying the full UNC path of a printer where the output should be printed. If empty, then the output will not be printed, though it may still be saved to a file.
filename	a *string* identifying the (server) directory path, name and extension to which the output should be saved. If empty, then the output will not be saved, though it may still be printed.

Returns:

command	uses Microsoft Word to perform a mail merge, using the specified *document* and data from the *folderID* in the Metastorm database, and sends the output to either a ***printer*** or a ***filename*** (in the server file system). You may specify the following combinations of ***printer*** or ***filename***: • neither - prints to default printer • **printer** - prints to nominated printer • **filename** - prints to nominated file • both **printer** and **filename** - prints to both nominated printer and file Microsoft Word must be installed on the same system as the Process Engine evaluating this function. The function executes asynchronously: the evaluation suite returns control to the Engine as soon as the request has been queued, not after it has been serviced.

For example:

%Print(%myDoc,`,"\\Domain1\VOL1\APPS\Support\Printers\HP4050",""`)

%Print("C:\My Documents\letter1.doc",%FolderID,"","C:\Merge.doc")

Data to be merged with a Word document must not contain non-standard ASCII characters.

The %Replace function can be used to ensure that non-standard ASCII characters such as " (ASCII 147) and " (ASCII 148) are not included in merge data.

%RaiseFlag(flag, folderID { , flag_data })

Arguments:

flag	a *string* identifying the flag to be raised. This should match a flag defined in one or more published procedures.
folderID	a *string* identifying the folder which should respond to this incidence of the flag. If empty, all folders waiting on the flag will be affected.
flag_data	any number of *strings*, comprising the individual data items (tab delimited) to be made available to each resultant action by way of the **`%Session.FlagData`** array variable.

Returns:

command	uses Metastorm BPM to raise the specified ***flag***, for any specified ***folderID***, and with any specified ***flag_data***. Any resultant actions execute asynchronously: the evaluation suite returns control to the engine as soon as the flag has been raised, not after any actions have been invoked or committed.

For example:

%RaiseFlag(myFlag,%FolderID)

%RaiseFlag(myFlag,"",%FolderName,%Subject**)**

%RegisterInQueue (symbolicName, system, eworkHost, queueSource, credentials, process, clientType, map, stage, action, params)

This function registers a new incoming queue definition in the Metastorm Engine. Additionally, the MQ Integration sub-components are informed of the new queue definition.

Arguments:

symbolicName	a *string* containing the name used to define this in queue.
system	a *string* specifying the messaging system used by the queue ("WebSphere MQ" or "MSMQ").
eworkHost	a *string* containing the parameters required to call the Metastorm Listener web service, acting on behalf of the target Process Engine. These parameters are a set of name-value pairs required to call the Metastorm Listener web service.

For details of these parameters, refer to Appendix D - Web Service Configuration Parameters.

queueSource	a *string* specifying the queue located on the MQ host.
credentials	a *string* containing the Metastorm user credentials to use when sending the TP eLoginRequest request. This is inserted as an XML field input list in the Login request.
process	a *string* containing the authentication process used when authenticating the user.
clientType	a *string* containing the client type passed to the Process Engine when authenticating the user.

 © Metastorm Inc., 2008

map	a *string* containing the map used when performing the TP action request.
stage	a *string* containing the name of the stage from which the action is performed. For a creation action, this is an empty string.
action	a *string* containing the action name used when performing the TP action request.
params	a *string* containing a set of custom parameters. This takes the form of name=value pairs, separated by semicolons. Parameters for JMS and Websphere MQ include: • retryNumber, which determines the number of times a message is resent to the MQ Listener's processMessage() interface if the MQ Listener is down or the host machine is not available. • retryInterval, which is the timeout value between retries. If the message continues to be sent unsuccessfully after this period, it is sent to the Retry queue. If a message fails because of an invalid Transaction Protocol request, it is sent to the Deadletter queue. The names of the Retry and Deadletter queues are stored in the eworkqueues.xml file. These are both Integers. If these values are not present in the properties string, a default value of 3 is used for retryNumber and 60 seconds for retryInterval.

Returns:

boolean	`TRUE` on success; otherwise, a Metastorm formula exception.

%RegisterOutQueue (symbolicName, system, queueName, params)

Arguments:

symbolicName	a *string* containing the name of the out queue.
system	a *string* containing the name of the messaging system used by the queue ("WebSphere MQ" or "MSMQ").
queueName	a *string* containing the name of the queue to which messages are sent.

params	a *string* containing a set of custom parameters.
	MSMQ queues support the following additional parameters within the *params* string.
	`PRIVATE = FALSE (default) || TRUE`
	`TRANSACTIONAL = FALSE (default) || TRUE`
	The private parameter indicates that the queue is private. Unlike public queues, the system cannot determine automatically whether the target queue is transactional. The transactional parameter is ignored for public queues but for a private queue it defines whether the message is sent within a transaction.
	A private queue will not return an error when sending messages to an invalid queue path. This is by design.

To register a WebSphere out queue enter the following:

```
URL=http://<APPSVR>:9080/eUMS/services/MessagingService?wsdl
```

where <APPSVR> is the name of the Application Server.

Returns:

boolean TRUE on success; otherwise, an Metastorm formula exception.

%Run(application, arguments)

Arguments:

application a *string* identifying the (server) directory path, name and extension of the application to be run. If the path is omitted, the server's Windows path will be used. If the extension is omitted, an .exe will be assumed.

arguments a *list* of arguments to pass to the application.

Returns:

command loads (from the server file system) and then runs the specified ***application***, with any specified ***arguments***.

The application in question must be installed on the same system as the Process Engine evaluating this function.

The function executes asynchronously: the evaluation suite returns control to the engine as soon as the request has been queued, not after it has been serviced.

For example:

```
%Run("C:\Program Files\my application\App.exe","/EW")
%Run("Notepad","")
```

%SaveData(folderID, filename, [Alt Delimiter])

Arguments:

folderID **must** be the *string* **%FolderID**.

filename a *string* identifying the (server) directory path, name and extension of a file to receive the folder data, as comma-delimited "quoted" values.

Alt Delimiter an optional *string* specifying the delimiter to use in place of a comma

Returns:

command saves all folder and custom variables, as (quoted) comma separated values, to the specified ***filename*** in the server file system.

 © Metastorm Inc., 2008

For example:

```
%SaveData(%FolderID,"C:\Windows\Temp\Folder.csv")
```

%SendMQMessage (symbolicName, message, priority, timeToLive, folderID)

Arguments:

symbolicName	a *string* containing the name of the registered out queue to use.
message	a *string* containing the message to send to the queue.
priority	an *integer* containing the priority of the message.
timeToLive	an *integer* containing the length time the message will remain in the queue. -1 indicates infinite time.
folderID	an optional *string* containing a folder ID for the message.

Returns:

boolean	TRUE on success; otherwise, an Metastorm formula exception.

%SendMQMessageEx (type, queueName, message, priority, timeToLive, params)

This function avoids the need to use a pre defined queue when sending a message.

Arguments:

type	a *string* containing the type of messaging system to use ("WebSphere MQ" or "MSMQ").
queueName	a *string* specifying the queue to which to send the message.
message	a *string* containing the message to send to the queue.
priority	an *integer* containing the priority of the message.
timeToLive	an *integer* containing the length of time the message will remain in the queue. -1 indicates infinite time.

params	a *string* containing a set of custom parameters. MSMQ queues support the following additional parameters in the *params* string.
	`PRIVATE = FALSE (default) | TRUE`
	`TRANSACTIONAL = FALSE (default) | TRUE`
	The private parameter indicates that the queue is private. Unlike public queues, the system cannot determine automatically whether the target queue is transactional. The transactional parameter is ignored for public queues but for a private queue it defines whether the message is sent within a transaction.
	A private queue will not return an error when sending messages to an invalid queue path. This is by design.

Returns:

boolean	TRUE on success; otherwise, an Metastorm formula exception.

%Script(language, file, procedure, map, statement [, args])

Arguments:

language	a *string* identifying the scripting language processor to be loaded (VBScript, JScript and JScript.NET are supported by the Designer).
file	if the specified ***language*** is VBScript or JScript, this is a *string* identifying the (server) directory path, name and extension of a script file to be loaded. If empty, no script will be loaded from file. If the specified **language** is JScript .NET, this is a list of files to be added to the script host when the assembly is compiled. These files will be looked at when the script is compiled. The script is compiled the first time it is used, on Engine start-up, and for every new version published thereafter.
procedure	if the specified ***language*** is VBScript or JScript, this is a *string* identifying a procedure holding a script attachment to be loaded. If empty, no procedure script will be loaded. If the specified ***language*** is JScript .NET, this is the name of the procedure script to be pulled in when compiling the assembly. This can be blank to indicate no procedure script.
map	if the specified ***language*** is VBScript or JScript, this is a *string* identifying a map holding a script attachment to be loaded. If empty, no map script will be loaded. If the specified ***language*** is JScript .NET, this is the name of the map script to be pulled in when compiling the assembly. This can be left blank to indicate no map script.
statement	if the specified ***language*** is VBScript or JScript, this is a *string* containing a statement, in the specified ***language***, to be executed. This will typically be a call to a function within one or other of the scripts loaded from the specified ***file***, ***procedure*** and/or ***map***. If any arguments need to be **"**quoted**"** (as string arguments do for VBScript), then each quotation mark must be preceded by a **%** character to prevent Metastorm 'stealing' it before it is passed to the script function. If the specified ***language*** is JScript .NET, this is a *string* identifying the name of a valid entry point in the map or procedure assembly that has been compiled.
arg	(JScript .NET only) any number of additional arguments to be packaged up and passed into the entry point's `args` argument.

Returns:

command	If the specified ***language*** is VBScript or JScript, this loads the specified ***language***, followed by any script in ***file*** (from the server file system), any specified ***procedure*** and/or ***map*** scripts (from the Metastorm database), and then executes the specified ***statement***.

 © Metastorm Inc., 2008

If the specified ***language*** is JScript .NET, the return value is True if the call is successful. Else the return value is False.

The evaluation suite returns control to the engine once the statement has been executed.

For example:

%Script(VBScript,"","",%MapName,"myFunc(%Priority)")

%Script(VBScript,"","",%MapName, "myFunc2(%"a string%",%"%Subject%")")

If you are using BizTalk's **Send a Document** Integration Wizard item:

%Script(VBScript, ,%Procedure.Name,%MapName, "SendDocument(%"PartnerName%", %"FormToSend%")")

%ScriptAsync(language, file, procedure, map, statement [, args])

Arguments:

language — a *string* identifying the scripting language processor to be loaded (VBScript, JScript and JScript.NET are supported by the Designer).

file — if the specified ***language*** is VBScript or JScript, this is a *string* identifying the (server) directory path, name and extension of a script file to be loaded. If empty, no script will be loaded from file.

If the specified ***language*** is JScript .NET, this is a list of files to be added to the script host when the assembly is compiled. These files will only be looked at when the script is compiled. This occurs the first time it is used, on Engine start-up and for every new version published thereafter.

procedure — if the specified ***language*** is VBScript or JScript, this is a *string* identifying a procedure holding a script attachment to be loaded. If empty, no procedure script will be loaded.

If the specified ***language*** is JScript .NET, this is the name of the procedure script to be pulled in when the assembly is compiled. This can be left blank to indicate no procedure script.

map — if the specified ***language*** is VBScript or JScript, this is a *string* identifying a map holding a script attachment to be loaded. If empty, no map script will be loaded.

If the specified ***language*** is JScript .NET, this is the name of the map script to be pulled in when this assembly is compiled. This can be left blank to indicate no map script.

statement — if the specified ***language*** is VBScript or JScript, this is a *string* containing a statement, in the specified ***language***, to be executed . This will typically be a call to a function within one or other of the scripts loaded from the specified ***file***, ***procedure*** and/or ***map***. If any arguments need to be **"**quoted**"** (as string arguments do for

VBScript), then each quotation mark must be preceded by a `%` character to prevent Metastorm 'stealing' it before it is passed to the script function.

If the specified ***language*** is JScript .NET, this is a *string* identifying the name of a valid entry point in the map or procedure assembly that has been compiled.

args (JScript .NET only) any number of additional arguments to be packaged up and passed into the point's `args` argument.

Returns:

command if the specified ***language*** is VBScript or JScript, this loads the specified ***language***, followed by any script in ***file*** (from the server file system), any specified ***procedure*** and/or ***map*** scripts (from the Metastorm database), and then executes the specified ***statement***.

If the specified ***language*** is JScript .NET, the return value is True if the system is able to queue this for asynchronous execution. Else the return value is False.

The function executes asynchronously: the evaluation suite returns control to the Engine as soon as the request has been queued, not after it has been serviced.

For example:

```
%ScriptAsync(VBScript,"C:\Scripts\Common Scripts.vbs","","",
    "myFunc(%Priority)")
```

%UnregisterInQueue (symbolicName)

Arguments:

symbolicName a *string* specifying the in queue to unregister.

Returns:

boolean -1 on success; otherwise, an Metastorm formula exception.

%UnregisterOutQueue (symbolicName)

Arguments:

symbolicName a *string* specifying the out queue to unregister.

Returns:

boolean -1 on success; otherwise, an Metastorm formula exception.

 © Metastorm Inc., 2008

%WriteText(string, filename ⟦ , new-file ⟧ saveasUnicode)

Arguments:

string	a *string* to be written to filename.
filename	a *string* identifying the (server) directory path, name and extension of the file to be written..
new-file	an optional *number* indicating whether any existing file should be replaced (1) or have string appended to it after a new line (0). If omitted, ***string*** will be appended to ***filename***, if it already exists.
saveAsUnicode	an optional *number* indicating whether the string data is stored as Unicode characters (1), or as ANSI (0) characters (0). The default is zero.

Returns:

command	saves ***string***, followed by a **`%NEWLINE()`**, to the specified ***filename*** in the server file system.

For example:

```
%WriteText("ipsem dolorem","C:\My Documents\Memo.txt")  // Append to Memo.txt, using
                                                        ANSI chars.
%WriteText(%Action.Notes,"C:\My Documents\Log.txt",1)       // Replace any current Log.txt


%WriteText(%StoryInRussianMemo,
                                "C:\My Documents\The Nose.txt",1,1)
                                // Writes out rather fine short story by Gogol in Unicode, replacing
                                anything currently in this file.
```

Send an e-mail message

This function is provided by the Mail Library (Mail.XEL).

It sends an email to the specified users, copying the specified ***cc*** and ***bcc*** lists, with the specified subject, ***message***, ***attachment*** and ***reply address***.

Arguments:

Recipient list	a *string* containing the To list for an email (delimited by semicolons).
CC list	an optional *string* containing the CC list for the email.
BCC list	an optional *string* containing the BCC list for the email.
Sender	a *string* containing the email address of the person who sent the message.
Subject	a *string* containing the subject for the email.
Message Body	a *string* containing the body of the email.
Body is HTML	a *boolean* expression indicating whether the message body is HTML.
Attachments	a string containing a (semi-colon) delimited list of file names.
Use SMTP Server	a *boolean* expression. A value of True indicates that an SMTP server should be used. False indicates that a pick-up folder should be used.
SMTP Server / Pickup Folder	a *string* containing the name of the SMTP server, or the folder from which email messaged are picked up for sending (depending on the value of the 'Use SMTP Server' parameter).
Port Number	If this optional *string* parameter is empty, or 0 then the default server port number specified in the machine configuration will be used.

Returns:

String	A string value indicating whether or not the function call was successful.

Operators

General Operators

These are used to assign values to variables, perform conditional evaluation, and alter the default order of evaluation for a formula. The following table lists the general operators:

Operator Name	*Operator Type*	*Function*
variable **:=** *formula*	*command*	Evaluates the *formula* and assigns its value to the *variable.*
condition **?** *formula* **:** *formula*	*any*	Evaluates the *condition;* if **TRUE**, evaluates the first *formula*, otherwise evaluates the second.
(*formula* **)**	*any*	Evaluates the parenthesized *formula* first. This may be used to alter the default order of evaluation in any conditional, logical, or arithmetic formula.

Table 44: General Operators

For example:

```
%Category := abc 123

%Category := %FolderName

( %FolderName = abc 123 )
        ? ( %Category := xyz 789 )
                : ( %Category := %EMPTY() )
```

Typecast Operators

These are used to convert data from one type to another. If the conversion cannot be performed, then evaluation of the property will resume on the next line or, if there are no further lines, fail. The following table lists the typecast operators:

Operator Name	*Operator Type*	*Function*
(boolean) *formula*	*Condition*	Attempts to convert *formula* to a *condition* (**TRUE** or **FALSE**) value.
(integer) *formula*	*Number*	Attempts to convert *formula* to a whole *number*.
(real) *formula*	*Number*	Attempts to convert *formula* to a real (floating point) *number*.
(date) *formula*	*Date-time*	Attempts to convert *formula* to a *date-time*.
(text) *formula*	*String*	Attempts to convert *formula* to a *string*.
(currency) *formula*	*Money*	Attempts to convert *formula* to a *money* data type.

Table 45: Typecast Operators

For example:

%myInteger := (integer) %myReal **// truncates real value**

Logical Operators

These may be used to construct complex logical formulas. Logical formulas are evaluated in the order listed below, with the most tightly binding shown first. The following table lists the logical operators:

Operator Name	Operator Type	Function
~ *condition*	*condition*	Changes **TRUE** to **FALSE** and vice versa.
formula **=** *formula*	*condition*	Returns **TRUE** if both *formulas* evaluate to the same value.
formula **>** *formula*	*condition*	Returns **TRUE** if the first *formula* evaluates a value greater than the second.
formula **<** *formula*	*condition*	Returns **TRUE** if the first *formula* evaluates a value less than the second.
formula **>=** *formula*	*condition*	Returns **TRUE** if the first *formula* evaluates a value greater than or equal to (but not less than) the second.
formula **<=** *formula*	*condition*	Returns **TRUE** if the first *formula* evaluates a value less than or equal to (but not greater than) the second.
formula **<>** *formula*	*condition*	Returns **TRUE** if the first *formula* evaluates a value not equal to the second.
condition **&** *condition*	*condition*	Returns **TRUE** only if both *conditions* are themselves **TRUE**.
condition **|** *condition*	*condition*	Returns **TRUE** if either *condition* is **TRUE**.

Table 46: Logical Operators

For example:

%myCheck := ~ (%Priority >= 4) **// TRUE if priority less than 3**

%myCheck := %myCheck & (%Priority <> 0) **// but not if zero**

Arithmetic Operators

These may be used to construct complex arithmetic formulas. Arithmetic formulas are evaluated in the order listed in the following table, with the most tightly binding shown first:

Operator Name	Operator Type	Function
- *number*	*number*	Changes a positive number to a negative, and vice versa.
number ***** *number*	*number*	Returns the result (product) of multiplying the two *numbers*.
number **/** *number*	*number*	Returns the result (quotient) of dividing the first *number* by the second.
number **+** *number*	*number*	Returns the result (sum) of adding the two *numbers*.
number **-** *number*	*number*	Returns the result (difference) of subtracting the second *number* from the first.

Table 47: Arithmetic Operators

 © Metastorm Inc., 2008

For example:

```
%myNumber := (integer) (%myNumber / 10 + 0.5)      // divide by 10 and round up
%myNumber := - %Abs(%myNumber)                     // force negative value
```

The arithmetic operators are also supported for Currencies. Adding, subtracting, multiplying or dividing two Currency values results in a Currency. Adding, subtracting or dividing a Currency and a Number results in a Real number. Multiplying a Currency and a Number results in a Currency.

Quotation marks

Although these are not generally required, as any string that is not recognized as a function, variable or operator will be treated as a text literal, there are certain circumstances where they are needed, such as when a string containing commas needs to be passed as a single argument to a function call.

When you are assigning a date time value to a variable, it must be enclosed in quotation marks. E.g. %DateTime1:="2001-11-23T00:00:00".

Quotes must be used in matching pairs. For instance, a typical SQL **SELECT** statement will itself include commas and 'single quoted' literal text values (Metastorm BPM treats 'single quotes' as literal values). In order to pass such a statement as an argument to a Metastorm function, the whole string should be **"**quoted**"** to prevent Metastorm BPM from trying to interpret the commas in the SQL statement itself as Metastorm function argument delimiters.

Note that putting any text in quotes does not prevent Metastorm from evaluating any Metastorm variables, functions or other operators within the quoted text.

For example:

```
%Category := abc 123                    // abc 123
%Category := "abc 123"                  // abc 123
%Category := 'abc 123'                  // 'abc 123'
%Category :=     abc 123                // abc 123 (loses leading spaces)
%Category := '   abc 123'               // '   abc 123' (keeps them)
%Category := "   abc 123"               // abc 123 (loses them)
%Subject := %Category                   // abc 123
%Subject := '%Category'                 // 'abc 123'
%Subject := "%Category"                 // abc 123
%Subject := %Category (xyz 789)         // abc 123 (xyz 789)
%Subject := %Category(xyz 789)          // UNRECOGNIZED FUNCTION !
```

%SelectSQL(SELECT * FROM eUser) // valid query

%SelectSQL (SELECT * FROM eUser) // UNRECOGNIZED VARIABLE !

%SelectSQL(SELECT eUserName,ePassword FROM eUser)
// 2 ARGUMENTS (BOTH INVALID) !

%SelectSQL('SELECT eUserName,ePassword FROM eUser')
// 2 ARGUMENTS (BOTH INVALID)!

%SelectSQL("SELECT eUserName,ePassword FROM eUser")
// valid query

%SelectSQL("SELECT eUserName,ePassword FROM eUser WHERE
eReportsTo = %User.Name") // INVALID SQL TEXT VALUE!

%SelectSQL("SELECT eUserName,ePassword FROM eUser WHERE
eReportsTo = '%User.Name'") // valid query

Escape operator

The **%** escape operator is used to force Metastorm BPM to treat the following character as itself. For instance, VBScript functions require string arguments to be quoted. Unless escaped, as in **%"the string%"**, Metastorm BPM will strip out the quotes before passing them to the function, which will then incorrectly assume the argument is numeric.

For example:

%Subject := %"abc *123***%"**	// "abc 123" (using % operator)
%Subject := %"%Category%"	// "abc 123"
%Subject := "%%Category"	// %Category
%Subject := %"%%Category%"	// "%Category"

Line separator

The **;** line separator is used to force the Metastorm Engine to discard any intermediate result (though not any side effects) and start evaluating the next line.

This allows multiple commands to be associated with a single property. It also allows commands to be included in **any** property, provided they are followed by a final formula of the appropriate type for that property.

Appendix C - Designing an Accessible Process

The Metastorm Web client is accessible to users with disabilities. This appendix describes points to consider when designing accessible procedures. The following accessibility criteria are supported and discussed:

- Section 508.
- Keyboard navigation.
- Internet Explorer text size.
- Color Blindness
- JAWS screen reader.

Section 508 Compliance

In 1998, American Congress updated section 508 of the Rehabilitation Act of 1973 to reflect a focus on technology. Section 508 aims to provide employees with disabilities access to the same systems and information as those without disabilities. Enforceable standards were put in place that define what makes a product considered accessible to people with disabilities.

For further information on Section 508 refer to www.section508.gov.

Watchfire's Bobby software runs a series of checks on software, to ensure that it is Section 508 compliant.

For further information about Watchfire® Bobby™ refer to http://bobby.watchfire.com.

No errors are highlighted when the Metastorm Web client is run through Bobby's Section 508 compliancy check. Bobby highlights 11 user checks. In order to pass Bobby's Section 508 compliancy test, Metastorm BPM addresses these user checks.

The process designer should take Section 508 into account when designing forms.

Keyboard Navigation

It is possible to navigate through most of the Web client using only the keyboard. The following are exceptions where a mouse must be used:

- Date picker.
- Time picker.

The date and time pickers are visual controls which can be used to select a date or time. Date and time fields can be completed using the keyboard.

We recommend that, for multiple selections from a list, you use two listboxes with an **Add** button and a **Remove** button, as the use of one listbox requires a mouse.

Using a keyboard to move through the items of a dropdown which has dependants causes each dependant field to be updated. We therefore recommend using a button to update the required field.

 © Metastorm Inc., 2008

Internet Explorer Text Size

Setting the text size in Internet Explorer affects the text in the Web client's list filter and To Do, Watch, Blank Forms or Admin Forms list, but does not affect form content. This is under the control of the process developer.

The process developer may set the font size of text in a Metastorm form using the options on the form's Property Editor **Format** tab. Doing this will affect each field's associated label, the text within each field and the size of each field. The process developer must set the font size of label fields separately, using the *Font* option on the label's Property Editor **Label** tab.

*If you use the **View Text Size** option to change the font size in the browser, the text in filter drop-downs remains the same size.*

Color Blindness

We recommend that, in order to accommodate color-blind users, you use '*' in the captions of required fields or choose backgrounds which can be distinguished from the red border by color-blind users.

JAWS for Windows Support

An unsighted user may use the Web client with the aid of the JAWS® screen reader. When designing procedures that will be accessed by JAWS users, be aware of the following:

Placing fields

When JAWS reads the contents of a form to the user, it reads the contents in the order in which the fields appear on the form (from the top left to the bottom right). You can aid the user by aligning each field caption to the left or top of the field. This will ensure that JAWS reads any useful information about the field before reaching the field itself.

Form fields

JAWS announces regular HTML field types to the user. It tells the user the field type, and in some cases provides instructions on how to populate the field.

In Metastorm BPM there are some complex field types that are made up of several HTML commands. JAWS does not know how to announce these Metastorm BPM specific fields. Make use of field captions and labels to help the user complete these fields.

Metastorm BPM specific fields include:

- Clips

The buttons on multiple attachment fields that allow you to add, open and delete attachments are JAWS links. To activate one of these links:

- *Press the Insert and F7 keys together to show the links list*
- *Scroll to the required link using the arrow keys*

- *Press the Enter key.*

  In order to read through the list of attachments, navigate to the field and press 'T' on the keyboard. Use the arrow keys to move between the cells of the table.

- Currencies
- Date times
- Grids
- Rules
- Numbers
- Signatures
- Status

Unsupported fields

Attachment clips that only allow a single attachment are not supported when using Metastorm BPM with JAWS. However, attachment clips that allow multiple attachments are supported.

Invalid Characters

If the user attempts to enter a letter into a Number or Currency field, JAWS reads out the letter but Metastorm BPM does not display the letter in the field. The user must tab out and back into the field to find out which characters have been successfully entered. It may be helpful to include a Label next to the field, which suggests that the user checks the contents of the field before submitting the form.

If the user enters an invalid character into a masked Edit field, a Number or a Currency field with the Min or Max property set, JAWS reads out the character, and Metastorm BPM displays it in the field and allows other characters to be entered. When the user tabs out of the field a message is displayed that states that the field data is invalid. JAWS reads out this message. The user will not be able to submit the form until the data in the field is valid.

Configuring JAWS

JAWS users will need to configure their JAWS keyboard options. JAWS uses the INSERT key as a 'modifier' key. INSERT is pressed in combination with other keys to activate commands.

Metastorm BPM uses the INSERT key to allow you to insert a row into an editable grid.

In order to be able to insert a row into an editable grid when running JAWS, configure JAWS to use just the numpad INSERT as the JAWS modifier key. This leaves the extended INSERT key free for using in an editable grid.

Accessing Metastorm BPM via JAWS

The following guidelines are intended to aid users accessing Metastorm BPM via the JAWS screen reader.

 © Metastorm Inc., 2008

Determining whether you are logged in to a service

If the list frame, 'eListsView frame', contains an Metastorm list, this indicates that you are logged in to a service. As you enter the frame, JAWS reads out the type of list (To Do, Watch, Blank Forms or Administration Forms) and the current page number of that list. For example, if the list is the first page of a Blank Forms list, JAWS reads out 'Metastorm Blank Forms page one'.

Determining the current list

If you are logged in to a service, a list is available in the 'eListsView frame'. When you navigate to the lists frame, JAWS reads out the list type. The list type is To Do, Watch, Blank Forms or Administration Forms.

To determine the filter that is currently applied to the list, navigate to the 'Metastorm filter' in the 'eFilterView frame' and traverse the filter tree items. JAWS reads out the following information for each filter item:

- Level in the filter tree
  - Level One – represents a service
  - Level Two – if the list is of Blank Forms or Admin Forms this represents the group that a form belongs to. If the list is of To Do or Watch items this represents the map a folder belongs to.
  - Level Three –represents the stage a folder is at. This is only available for a To Do or Watch list.
- Filter item text
- Selected (only included if item is currently selected)
- Open/closed
- Number of siblings or children

The following are examples of the text read out by JAWS as the filter tree is traversed:

- 'Link Level One, My Metastorm BPM Service, Closed, One of One'. This is read out for the service item, My Metastorm Service. It is the only service available and the filter item is not open.
- 'Link Level One, My Metastorm Service, Selected, Open, One item'. This is read out for the service item, My Metastorm Service. It is the currently selected filter item, is open and contains one sub item.

JAWS reads out 'Selected' for the currently selected filter item.

If the current list is a To Do list, and the selected tree item is a map item, the current list includes all folders for that map.

Logging in to a service

To log in to a service, navigate to the 'Metastorm filter', and to the service you wish to log in to. The 'Level One' filter items represent services. Press the Enter key when you reach the service you wish to log in to. The resulting dialog allows you to submit your user name and password.

Navigating to the submit and cancel buttons on a form

To submit or cancel a form, you may need to use CTRL+TAB to navigate to the button frame, and then the TAB key to navigate to the buttons. Alternatively, you can press **G** to access the buttons.

For further information on keyboard navigation of the Web Client refer to the Using Metastorm BPM with Internet Explorer document.

 © Metastorm Inc., 2008

Appendix D – Business Rules Web Service Configuration Parameters

The Business Rules web service requires a configuration string in the form of a set of name-value pairs. The following named items are supported.

Connection Parameters

URL – string

Summary

Sets the base URL of the XML Web service the client is requesting.

Remarks

Timeout – long

Summary

Indicates the time an XML Web service client waits for a synchronous XML Web service request to complete (in milliseconds).

Remarks

Setting the Timeout property to -1 indicates that the request does not time out. Even though an XML Web service client can set the Timeout property to not time out, the Web server can still cause the request to time out on the server side.

Authentication Parameters

AuthMethod – long

Summary

Sets the base URL of the XML Web service the client is requesting.

Remarks

The following authentication mechanisms are supported:

0 – Anonymous

1 – Basic Authentication (requires AuthUser, AuthDomain and AuthPassword)

2 – Integrated Windows Authentication

AuthUser – string

Summary

Sets the username used for basic authentication.

Remarks

AuthDomain – string

Summary

Sets the domain used for basic authentication.

Remarks

AuthPassword – string

Summary

Sets the password used for basic authentication.

Remarks

Remember that password information is sent in clear text over the network when using basic authentication.

Proxy Properties

ProxyServer – string

Summary

Sets the information for the proxy server used for all web services calls.

Remarks

The proxy server can be configured in two ways:

CURRENT_USER	The current users I.E. settings are used to determine the proxy
host:port	The proxy at host:port is used (e.g. http://93.13.17.2:8080)

ProxyUser – string

Summary

Sets the username used when accessing the proxy server.

 © Metastorm Inc., 2008

Remarks

ProxyPassword – string

Summary

Sets the password used when accessing the proxy server.

Remarks

BypassProxyOnLocal – boolean

Summary

Determines whether the proxy server is bypassed when calling local resources.

Remarks

If BypassProxyOnLocal is true, requests to local Internet resources do not use the proxy server. Local requests are identified by the lack of a period (.) in the URI, as in http://webserver/ or access the local server, including http://localhost, http://loopback, or http://127.0.0.1 . When BypassProxyOnLocal is false, all Internet requests are made through the proxy server.

Advanced Properties

AllowAutoRedirect – boolean

Summary

Determines whether the client will automatically follow server sided redirects.

Remarks

If a client sends authentication information, such as a user name and password, you do not want to enable the server to redirect, because this can compromise security.

KeepAlive – boolean

Summary

Sets the Connection: KeepAlive header in the HTTP requests.

Remarks

If the KeepAlive property is set to true the Connection: KeepAlive header will be set. This means SOAP calls will be made through the same HTTP connection when available.

The Connection: KeepAlive header can be overridden by the IIS server

UserAgent – string

Summary

Sets the user agent string for SOAP calls.

Remarks

The user agent string allows a Web server to identify the client.

PreAuthenticate – boolean

Summary

Sets the base URL of the XML Web service the client is requesting.

Remarks

When PreAuthenticate is true, the WWW-authenticate header is sent with the first request if the authentication mechanism supports doing so. When PreAuthenticate is false, a request is made to the XML Web service method without initially attempting to authenticate the user. If the XML Web service allows anonymous access, then the XML Web service method is executed. If anonymous access is disallowed, a 401 HTTP return code is sent back to the client. In response, authentication credentials are returned to the Web server. If the client is authenticated and subsequently authorized to access the XML Web service, the XML Web service method is executed; otherwise the client is denied access.

RequestEncoding – long

Summary

Sets the base URL of the XML Web service the client is requesting.

Remarks

The RequestEncoding determines the encoding for the request message. The ContentType of the request will be annotated with the encoding value.

0 - System.Text.ASCIIEncoding
1 - System.Text.UnicodeEncoding
2 - System.Text.UTF7Encoding
3 - System.Text.UTF8Encoding

 © Metastorm Inc., 2008

Appendix E – Masks Format

Masks are defined in *.DXM files. The contents of an example DXM file are shown below:

```
[RegExpr1]
ShortDescription=Phone
Description=4155551212
Mask=!\(999\)000-0000;1;_

[RegExpr2]
ShortDescription=Extension
Description=15450
Mask=!99999;1;_

[RegExpr3]
ShortDescription=Time of day
Description=15:25
Mask=(0?\d | 1\d | 2[0-3]) : [0-5]\d

[RegExpr4]
ShortDescription=Post Code
Description=SW19 7AB
Mask= [a-z]{1,2}\d{1,2}[a-z]?\s*\d[a-z]{2}

[RegExpr5]
ShortDescription=Email
Description=support@metastorm.com
Mask=[-a-z0-9_.]+@([-a-z0-9]+\.)+[a-z]{2,6}
```

Table 48: Example DXM File

DXM files have the format of *.INI files.

Each DXM file defines several masks and is divided into a number of sections, one for each mask. Each section must:

- Have a name containing one of the following keywords:
  - **Standard**.
  - **RegExpr**.

- Contain the following keys:
  - **ShortDescription**.
  - **Description**.
  - **Mask**.

 © Metastorm Inc., 2008